

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ РОБОТИ З ХМАРНИМИ
БАЗАМИ ДАНИХ НА ПРИКЛАДІ ПРОГРАМИ ДЛЯ
ПАЦІЄНТІВ З ЦУКРОВИМ ДІАБЕТОМ
RESEARCH OF METHODS AND TOOLS FOR WORKING WITH CLOUD
DATABASES AS ILLUSTRATED BY THE APPLICATION
FOR PATIENTS WITH DIABETES

Виконала: студентка 2 курсу, групи КНТ-120м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Колпакова Д.В.

(прізвище та ініціали)

Керівник Табунщик Г.В.

(прізвище та ініціали)

Рецензент Шитікова О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ПРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
 (КОД І НАЙМЕНУВАННЯ)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20__ року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Колпаковій Дар'ї Валеріївні

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження методів та засобів роботи з хмарними базами даних на прикладі програми для пацієнтів з цукровим діабетом.
Research of methods and tools for working with cloud databases as illustrated by the application for patients with diabetes.

керівник проекту (роботи) Табунщик Галіна Володимирівна, к.т.н., професор
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “ 05 ” листопада 2021 року № 435

2. Строк подання студентом проекту (роботи) 13 грудня 2021 року

3. Вихідні дані до проекту (роботи) рекомендована література

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Розробка архітектури програми. 4. Розробка програми. 5. Експлуатація, тестування та експериментальне дослідження програми. 6. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Табунщик Г.В., професор		
Нормоконтролер	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання “ 27 ” вересня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4–7 тижні	Розділ 2
4	Розробка архітектури програми.	8 тиждень	Розділ 3
5	Розробка програми.	9–10 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	11 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	12-13 тиждень	Додатки
8	Нормоконтроль та рецензування.	14–15 тижні	
9	Захист роботи.	16 тиждень	

Студентка

(підпис)

Колпакова Д.В.

(прізвище та ініціали)

Керівник проекту (роботи)

(підпис)

Табунщик Г.В.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
130 с., 6 табл., 69 рис., 2 дод., 37 джерел.

ХМАРНІ БАЗИ ДАНИХ, FIREBASE, CLOUD FIRESTORE, AWS AMPLIFY, BACKENDLESS, МОБІЛЬНИЙ ЗАСТОСУНОК, FLUTTER, DART, ПАЦІЄНТ, ЛІКАР, ЦУКРОВИЙ ДІАБЕТ.

Об'єкт дослідження – хмарні бази даних.

Предмет дослідження – методи та засоби роботи з хмарними базами даних; взаємодія хмарних баз даних з програмними засобами.

Мета роботи – покращення процесу обслуговування пацієнтів з цукровим діабетом за рахунок використання хмарної бази даних з мобільним застосунком.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, хмарні бази даних: Cloud Firestore від Firebase, бази даних від AWS Amplify та бази даних від Backendless; мова програмування Dart, програмний каркас Flutter, середовище розробки Android Studio, персональний комп'ютер з процесором Intel Core I7 під управлінням операційної системи MacOS Big Sur.

Результати. Проведено дослідження існуючих хмарних баз даних. Створено мобільний застосунок для пацієнтів з цукровим діабетом та лікарів за допомогою програмного каркасу Flutter з використанням методів та засобів хмарної бази даних Cloud Firestore.

Галузь використання – медицина, медичні заклади.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 130 pages, 69 figures, 6 tables, 2 appendixes, 37 sources.

CLOUD DATABASES, FIREBASE, CLOUD FIRESTORE, AWS AMPLIFY, BACKENDLESS, MOBILE APPLICATION, FLUTTER, DART, PATIENT, DOCTOR, DIABETES.

Object of study is cloud databases.

Subject of study are methods and tools for working with cloud databases; interaction of cloud databases into software.

The purpose of the work is to improve the process of caring for patients with diabetes through the use of a cloud database with a mobile application.

Materials, methods and tools: object-oriented programming, cloud databases: Cloud Firestore by Firebase, databases by AWS Amplify, databases by Backendless; programming language Dart, software framework Flutter, integrated development environment Android Studio, personal computer with an Intel Core i7 processor under the control of the MacOS Big Sur operating system.

Results. A study of existing cloud databases was conducted. A mobile application has been created for patients with diabetes and their doctors using the software framework Flutter and the methods and tools of the cloud database Cloud Firestore by Firebase.

Field of use are medicine, medical institutions.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз проблеми та постановка завдань дослідження	11
1.1 Короткий опис досліджуваної предметної галузі	11
1.2 Огляд існуючих серверних провайдерів	13
1.2.1 Firebase	13
1.2.2 AWS Amplify	16
1.2.3 Backendless	19
1.3 Порівняльний аналіз серверних постачальників послуг	21
1.4 Короткий опис стану хворих на цукровий діабет та аналіз програмних рішень для них	23
1.4.1 Аналіз існуючих програмних продуктів	25
1.5 Створення бізнес-моделі на основі отриманих даних	29
1.6 Висновок до першого розділу	32
2 Матеріали і методи	33
2.1 Методи роботи систем управління базами даних	33
2.2 Методи впровадження хмарних сервісів у програмне забезпечення	36
2.3 Висновок до другого розділу	41
3 Розробка архітектури програми	42
3.1 Бізнес-логіка програмного продукту	42
3.2 Архітектура програмного продукту	44
3.3 Висновок до третього розділу	49
4 Розробка програми	50
4.1 Обґрунтування вибору програмного каркасу для розробки програми .	50
4.2 Особливості та вимоги для застосунку створеному за допомогою програмного каркасу Flutter	54
4.3 Вибір середовища розробки	56
4.5 Налаштування серверного постачальника послуг	58

	7
4.5 Розробка програмного продукту	62
4.6 Висновок до четвертого розділу	69
5 Експлуатація, тестування та експериментальне дослідження програми	71
5.1 Тестування Cloud Firestore	71
5.2 Тестування взаємодії застосунку і сервісів Firebase.....	75
5.3 Огляд інтерфейсу користувача розробленого застосунку	79
5.4 Модульне тестування програми	86
5.5 Висновок до п'ятого розділу.....	87
Висновки.....	89
Перелік джерел посилання	91
Додаток А Текст програми	95
Додаток Б Слайди презентації.....	119

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AI	– Artificial Intelligence;
API	– Application Programming Interface;
BaaS	– Backend as a Service;
BLoC	– Business Logic Components;
CLI	– Command-line interface;
DTO	– Data Transfer Object;
IDE	– Integrated Development Environment;
MBaaS	– Mobile Backend as a Service;
SDK	– Software Development Kit;
UI	– User Interface;
URL	– Uniform Resource Locator;
UML	– Unified Modeling Language;
БД	– база даних;
ООП	– об’єктно-орієнтоване програмування;
ОС	– операційна система;
ПЗ	– програмне забезпечення;
ЦД	– цукровий діабет.

ВСТУП

На сьогодні важко уявити мобільний застосунок який не має доступу до мережі Інтернет та не зберігає дані у віддалену базу даних. Останнім часом, для розробки застосунків, компанії вирішують відмовитися від серверної розробки та віддають перевагу BaaS (бекенд як сервіс).

Актуальність даної теми полягає у тому, що використання сервісів, які пропонують доступ до хмарних баз даних, а також авторизацію, аналітику тощо, є розповсюдженим явищем бо такі ресурси надають можливість не залежати від бекенд розробки, отже зменшити витрати на розробку продукту. Існує безліч таких платформ, їх розробили та постійно вдосконалюють такі провідні корпорації у IT-індустрії як: Google, Amazon, Microsoft, Facebook та інші. Саме тому, в умовах високої конкуренції, важливо вивчити існуючі рішення від глобальних та маленьких компаній, щоб в подальшому, знайти оптимальний сервіс який буде відповідати вимогам проєкту.

Об'єктом дослідження є хмарні бази даних та сервіси, які пропонують доступ до них.

Предмет дослідження – методи та засоби роботи з хмарними базами даних та взаємодія хмарних баз даних з програмними засобами.

Головна мета роботи – розробити мобільний застосунок для пацієнтів з цукровим діабетом та лікарів, та впровадити відповідний сервіс, що надає доступ до хмарної бази даних.

Для досягнення поставленої мети у роботі, слід вирішити наступні завдання:

- а) проаналізувати існуючі BaaS або MBaaS (мобільний бекенд як сервіс), знайти та вивчити переваги та недоліки кожної платформи;
- б) визначити який сервіс слід впровадити згідно вимогам проєкту що розробляється;
- в) вивчити стан проблеми захворювання на цукровий діабет в Україні та в світі;

- г) проаналізувати існуючі програмні рішення для людей хворих на діабет;
- д) скласти бізнес модель програмного продукту, що розробляється на основі отриманих статистичних даних;
- е) розробити архітектуру для впровадження взаємодії застосунку з хмарною базою даних;
- є) розробити програму зі запропонованою архітектурою;
- ж) протестувати продуктивність та працездатність застосунку обраного сервісу, який впроваджено у програмний продукт.

Вибір тематики програмного продукту обумовлений тим, що існує безліч застосунків для людей хворих на цукровий діабет, але вони мають надто складний інтерфейс. Передбачається що головна функція програмного рішення буде полягати у збереженні статистичних даних про стан пацієнта у хмарну базу даних та у наданні можливості лікарям спостерігати за станом їх пацієнтів.

У роботі буде розглянуто такі бекенд-сервіси як: Firebase (Google), AWS Amplify (Amazon) та Backendless, та проаналізовано хмарні бази даних які вони надають.

Розробка мобільного застосунку буде здійснюватися за допомогою об'єктно-орієнтованого програмування з використанням мови програмування Dart, програмного каркасу Flutter, додаткових бібліотек, для спрощення реалізації програмного рішення, та середовища розробки Android Studio. Основним технічним засобом є персональний комп'ютер з процесором Intel Core I7 під управлінням операційної системи MacOS Big Sur.

Часткові результати роботи, яку було подано у вигляді тези на науково-практичну конференцію викладачів, науковців, молодих учених, аспірантів та студентів “Тиждень науки – 2021”, було використано в даній дипломній кваліфікаційній роботі.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Короткий опис досліджуваної предметної галузі

Мобільний бекенд як сервіс або скорочено MBaaS (також часто називають BaaS) – це архітектура сервісу хмарних обчислень, яка діє як проміжне програмне забезпечення, що дозволяє розробникам пов'язувати свої мобільні застосунки з хмарними сервісами через API та комплекти розробки програмного забезпечення (SDK). Вона дозволяє розробникам автоматизувати функції серверної розробки та забезпечити якісне управління хмарної інфраструктури, прискорює завдання розробки та допомагає компаніям краще реагувати на запити користувачів. Постачальник послуг BaaS відповідає за роботу серверів і виконання технічного обслуговування, а сам бекенд-сервіс надає розробникам інструменти, які допомагають їм створювати бекенд-код, що прискорює розробку. Часто функції BaaS включають API, сховище файлів, управління даними, push-повідомлення тощо. MBaaS і BaaS принципово схожі, з тією лише різницею, що BaaS більш орієнтовні на створення веб-застосунків [1].

Основні переваги бекенд-сервісу [2]:

а) масштабованість. Використання такої платформи є вигідним, оскільки така модель пропонує розробникам велику масштабованість. Як результат, користувачі, які вибирають платформу BaaS, можуть скористатися великою кількістю послуг та функцій, як з безкоштовними, так і з платними опціями. Це робить масштабування застосунків, заснованих на зростаючих потребах, досить легким та полегшує управління ресурсами.

б) простота в експлуатації. BaaS – це безпроблемна модель хмарного сервісу, оскільки вона не вимагає від розробників виконання будь-якого хостингу або обслуговування. Постачальник послуг BaaS здійснює всі ці дії за замовченням. Таким чином розробники організовано та швидко створюють застосунки за рахунок спрощення бекенд-розробки. Це дозволяє їм приділяти особливу увагу поліпшенню інтерфейсу користувача у застосунку. Таким

чином, компанія-розробник може запропонувати користувачам зручний застосунок, з більш продуманою бізнес-логікою.

в) економія часу. Використання бекенд-сервісу прискорює розробку, оскільки немає необхідності запускати хмарні сервери та керувати ними. Легкодоступні ресурси в BaaS дозволяють зручно виконувати ключові дії при розробці.

г) зниження витрат на розробку. Використання BaaS знижує витрати на розробку, так як відпадає необхідність платити бекенд-розробникам за розробку серверної частини. Витрати на розробку досить низькі при використанні рішення BaaS. За допомогою бекенд-сервісів можна значно заощадити на інфраструктурних витратах. Компанії, які покладаються на бекенд-сервіси, не потребують послуг ІТ-фахівців для виконання різних завдань, пов'язаних з бекенд-розробкою.

Бекенд-сервіс вважається гнучкою і оптимальною моделлю хмарного сервісу та допомагає спростити створення API. Таке API розроблюється в першу чергу, використовується для підключення застосунків до хмарних серверних служб та функціонує як основа для взаємодії серверу з платформами iOS або Android.

API або інтерфейс прикладного програмування можна визначити як опис способів, якими одна комп'ютерна програма може взаємодіяти з іншою. Щоб зрозуміти API, можна взяти приклад транспортного засобу, API якого матиме інформацію про його властивості, тобто гальмуванні, прискоренні, зміни тощо. API також матиме інформацію про те, як ці функції будуть виконуватися транспортним засобом. Отже якщо провести аналогію з програмою: транспортний засіб – це сервер, API – елементи керування транспортом (наприклад кермо) та водій як застосунок [3, с. 56].

API надає такі можливості як [3, с. 60]:

а) автоматизація. Вона скорочує кількість завдань, які необхідно виконувати вручну, автоматизуючи їх. Використання API мобільного бекенда – це відмінний спосіб для розробників швидше керувати робочими процесами

та впроваджувати оновлення для підвищення продуктивності без затримок. Компанії і розробники покладаються на API, щоб швидше вирішувати завдання і скорочувати кількість необхідних зусиль.

б) зручність. Використання надійної API мобільного бекенд платформи підвищує можливості бізнесу справлятися зі зростаючим попитом. Реалізація багатофункціонального API допомагає швидко розподіляти інформацію з різних систем і без особливих зусиль запускати нові сервіси для клієнтів. API також дозволяють розробникам створювати призначені для користувача служби для нових груп користувачів. Впровадження API в робочий процес бізнесу допомагає швидко створювати системні зміни без затримок.

в) ефективність. Просувати новий контент може бути досить простим, якщо для цього використовують правильне API. Набагато зручніше публікувати контент і автоматично робити його доступним по всіх каналах. Багато компаній покладаються на API для швидкого донесення свого контенту до великої аудиторії і можуть допомогти їм уникнути більш дорогих і трудомістких варіантів традиційного маркетингу. Використання API може бути відмінним способом підвищення ефективності публікацій та маркетингу.

Так як планується розробка мобільного застосунку за допомогою програмного каркасу Flutter, слід проаналізувати саме ті серверні платформи послуг, які підтримують взаємодію з Flutter SDK. Слід зауважити, що якщо бекенд-сервіс на пряму не підтримує обраний SDK, тоді важко, а іноді неможливо, власноруч організувати взаємодію між сервером та застосунком.

1.2 Огляд існуючих серверних провайдерів

1.2.1 Firebase

Firebase (див. рис. 1.1) популярна платформа BaaS від Google, яка надає широкий спектр функцій і компонентів, що допомагають в розробці мобільних і веб-застосунків.



Рисунок 1.1 – Логотип Firebase [4]

Firebase була розроблена Envolvе, стартап-компанією, заснованої у 2011 році. Компанія запропонувала API для розробників, щоб полегшити інтеграцію онлайн-чату для веб-сайтів. Але засновники Envolvе виявили, що їх сервіс чату використовується розробниками як платформа для синхронізації даних програм у реальному часі. Тому було вирішено розділити архітектуру реального часу від системи чату, що призвело до заснування Firebase у 2011 році. Платформа була публічно запущена у квітні 2012 року. Першим запущеним продуктом Firebase була база даних Firebase Realtime Database. Firebase став частиною Google у жовтні 2014 року. У той же час компанія запустила Firebase Authentication та Firebase Hosting, завдяки чому стала провідним MBaaS.

Далі розглянемо переваги використання Firebase. По перше, Firebase безкоштовний для початку та дозволяє користувачам авторизуватись у системі за допомогою свого облікового запису Google. План Spark для Firebase є безкоштовним і пропонує багато функцій, які допоможуть розробникам розпочати роботу. На основі зростаючих вимог вони можуть вибрати план Blaze (див. рис. 1.2), який враховує початкові безкоштовні обмеження. Наприклад, припустимо, що розробник розмістить 20 ГБ пам'яті на плані Blaze. Безкоштовні квоти охоплюють 10 ГБ, і це означає, що Firebase стягне плату лише за інші 10 ГБ, які були використані додатково.

Також Firebase може допомогти розробникам скоротити час виходу продукту на ринок. Зазвичай кожному розробнику потрібен доступ до сервера та хосту для створення та обслуговування бази даних та серверних служб.

Тому для створення застосунків необхідно мати бекенд-розробника та розробника інтерфейсу. Це справедливо навіть для невеликих застосунків, де бекенд і фронтенд повинні співпрацювати на різних етапах.

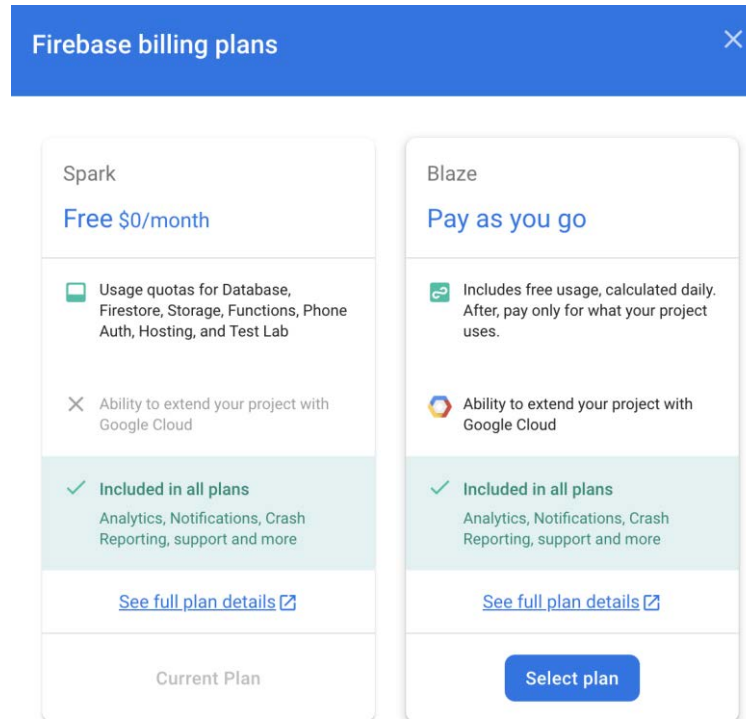


Рисунок 1.2 – Тарифні плани Firebase [4]

Наявність розробників інтерфейсу та серверної частини часто може призвести до помилок та проблем, які можуть спричинити проблеми з застосунком, що збільшує вартість та складність розробки. Однак використання Firebase дозволяє розробникам інтерфейсу керувати сервісами платформи та скоротити час, необхідний для завершення розробки. Це здійснюється за допомогою готових до використання сервісів для зберігання даних, авторизації, сповіщень, аналітики тощо. А завдяки високоякісним пакетам SDK, детальній документації та активній спільноті розробників інтеграція Firebase у кодову базу проєкту не займає багато часу.

Далі однією з переваг є те, що Firebase надає розробникам повний список продуктів, які допоможуть їм у процесі розробки [4].

Платформа надає два варіанти бази даних: Cloud Firestore та Realtime Database. Також Firebase дозволяє без зусиль виконувати хмарне зберігання медіа та дозволяє розробляти застосунки без сервера за допомогою інтегрованих хмарних функцій.

Firebase охоплює весь цикл розробки застосунків, платформа містить функції для створення, випуску та моніторингу програм. Крім того, як останній крок циклу розробки застосунків, платформа надає інструменти для залучення користувачів та продовження використання ними розроблених застосунків.

Слід також зауважити, що Firebase працює під керівництвом однієї з найвідоміших і найнадійніших корпорацій у світі технологій. З моменту свого придбання компанією Google Firebase, Google активно інвестує в платформу та вдосконалює свій продукт [4].

Проте, Firebase також має недоліки. Основним є те, що користувачі платформи не можуть змінювати вихідний код Firebase, оскільки він закритий. Багато розробників, яким подобаються функції Firebase, в кінцевому підсумку використовують інші платформи через це обмеження. Незважаючи на те, що платформа не має відкритого вихідного коду, важливо підкреслити, що багато бібліотек та SDK доступні на GitHub.

Ще одним недоліком можна вважати те, що бази даних Firebase, Cloud Firestore та Firebase Realtime Database, пропонують структуру NoSQL, що не надає можливості використовувати реляційну базу даних. Це породжує ще одну проблему, а саме питання реалізації складних запитів: проблеми пов'язані з фільтрацією, поєднанням фільтрів та комбінуванням кількох запитів.

1.2.2 AWS Amplify

AWS Amplify (див. рис. 1.3) – це повноцінна платформа, яка була розроблена, щоб допомогти розробникам створити повномасштабні програми,

якими керує Amazon Web Services. Платформа поставляється з безліччю інструментів і служб, які дозволяють користувачам налаштовувати серверні програми, підключати застосунки тощо [5].



Рисунок 1.3 – Логотип AWS Amplify [5]

AWS Amplify був запущений у 2017 році. Найважливіша перевага цього інструменту полягає в тому, що він дозволяє безпечно інтегрувати безліч функцій для всього, від API до AI.

Як і Firebase, AWS Amplify має як переваги, так і недоліки. Головною перевагою платформи є те, що вона забезпечує простий, швидкий та сучасний підхід до створення мобільних та веб-застосунків, який керується інтерфейсом користувача. Готовий компонент інтерфейсу користувача має все необхідне, тому розробникам не потрібно його кодувати. Проектування процесів і робочих процесів у CLI також є безпроблемним, що прискорює розробку застосунків.

Також платформа пропонує достатньо систем для управління БД, що дозволяє розробникам вибрати такий тип, який для них є найбільш зручним (див. рис. 1.4).

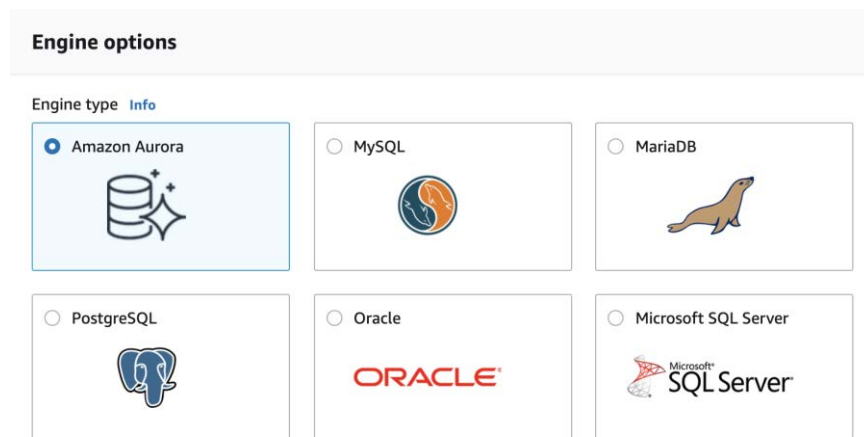


Рисунок 1.4 – Бази даних з якими працює AWS Amplify [5]

Як і багато інших платних служб, модель оплати для AWS Amplify дуже гнучка та економічно ефективна, оскільки плата стягується лише за послуги, якими користуються. Однак, при реєстрації, платформа вимагає прив'язати банківську карту, що може бути не зручним.

AWS Amplify має два тарифних плани (див. рис. 1.5):

а) Free Tier. Один із перших способів оплати, який також можна назвати пробним періодом у 12 місяців.

б) Pay As You Go. Після перших 12 місяців безкоштовний рівень перетворюється на модель Pay As You Go, що означає, що плата стягується за все, що використовується застосунком.

До недоліків можна віднести вищу криву навчання у порівнянні з іншими платформами, тобто новачкам треба витратити багато часу на вивчення можливостей цієї платформи.

Static Web Hosting - Free Tier	Static Web Hosting - Pay as you go
Build & Deploy	Build & Deploy
1000 build minutes per month	\$0.01 per build minute
HOSTING	HOSTING
5 GB stored per month	\$0.023 per GB stored per month
15 GB served per month	\$0.15 per GB served
<i>The free tier expires at the end of your 12 month AWS Free Tier term.</i>	<i>Multiple sites per project, public SSL certificates included at no extra cost</i>

Рисунок 1.5 – Тарифні плани AWS Amplify [5]

Також платформа постійно змінюється, нові функції та сервіси постійно впроваджуються, що означає, що користувач AWS Amplify повинен завжди бути в курсі подій на платформі. Тобто мало того що, на вивчення платформи потрібен час, так ще постійно треба не забувати моніторити зміни AWS. Як результат, це може ускладнити розробку.

1.2.3 Backendless

Backendless (див. рис. 1.6) – це ефективна платформа для розробки застосунків, яка була розроблена для обслуговування окремих розробників, а також агентств.



Рисунок 1.6 – Логотип Backendless [6]

Backendless був заснований в 2012 році Марком Піллером. Розробка розпочалася 1 серпня 2012 року. Відкрита бета-версія Backendless запущена 5 березня 2013 року з версією платформи 1.0. Початкові функції включали керування користувачами, реляційні дані, push-сповіщення, повідомлення про публікацію/підписку та геолокацію.

Це одне з найкращих MBaaS з відкритим вихідним кодом, яке принесло повний набір інструментів, щоб допомогти розробникам зробити їх завдання розробки бекенда простішим, ефективнішим та зручнішим. Професійна версія Backendless може дозволити розробникам запускати свої служби на власних серверах. Платформа також надає розробникам сервіси, які необхідні для створення бекенда. Це включає у себе управління розгортанням, моделювання даних, мобільну розробку, налагодження та контроль джерел – усе в одному місці. Платформа пропонує основні API, такі як база даних, файлобмін, геолокація, обмін повідомленнями тощо [6].

Зручно те, що в Backendless кожен API, який потрібен для програми, буде створено цим програмним забезпеченням автоматично. Розробники можуть легко керувати цими автоматично згенерованими API та використовувати їх у своїх проєктах розробки. Автоматичне створення API – це та функція, яка виділяє Backendless серед конкурентів.

Платформа пропонує масштабовану, динамічну та безпечну базу даних SQL і NoSQL власної розробки. Функції безпеки бази даних дозволять визначати дозволи для ролей користувачів графічно. За допомогою Backendless генераторів коду можна миттєво завантажити клієнтський код, зіставлений зі схемою бази даних [6].

Також Backendless дозволяє побудувати логіку коду візуально. При цьому API код не потрібен. Створення та тестування безкодових логік на консолі може допомогти повторно використовувати ці логіки в будь-якому місці, в будь-який час.

Цікаво те що, для початківців платформа пропонує курс знайомства з платформою у вигляді гри. Протягом проходження цього курсу користувачі отримують бали або, як платформа їх називає, – Backendless Bucks (BB). Це так звана валюта яку можна або купити, або заробити. Як показано на рисунку 1.7, щоб отримати безкоштовний тарифний план, треба заробити 15000 BB, або $15000 / 300 = 50$ доларів США (1 долар США дорівнює 300 BB). Слід зауважити, що безкоштовний тарифний план не надає можливостей для реальних проєктів, її можна порівняти з пісочницею для того, щоб вивчити основні можливості платформи.

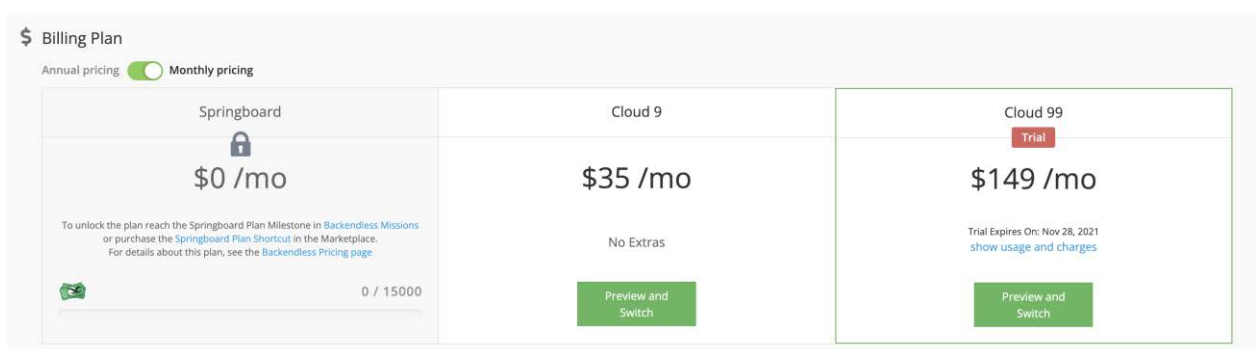


Рисунок 1.7 – Тарифні плани Backendless [6]

Також платформа пропонує платні плани, які можуть бути активовані лише за реальну валюту: Cloud 9 та Cloud 99. Різниця між ними лиш у тому, що Cloud 99 має набагато вищі квоти користування ніж Cloud 9.

1.3 Порівняльний аналіз серверних постачальників послуг

У попередньому підрозділі було розглянуто 3 платформи які пропонують бекенд як сервіс: Firebase, AWS Amplify та Backendless. Усі вони мають переваги та недоліки, тому наступне завдання полягає у тому, щоб знайти оптимальне рішення для розробки запропонованого програмного рішення, а також, використовуючи обраний сервіс, вдосконалити впровадження платформи у застосунок. Далі наведено таблицю 1.1, для порівняння характеристик розглянутих BaaS. Це лише основні показники, які наочно показують збіг або різницю між сервісами.

Таблиця 1.1 – Порівняльна таблиця серверних платформ

Показники	Firebase	AWS Amplify	Backendless
Рік заснування	2011	2017	2012
Бази даних	Cloud Firestore та Realtime Database (NoSQL)	SQL і NoSQL від відомих систем управління управління БД	SQL і NoSQL власної розробки, зі зручним UI
Рівень складності	Середній, наявність активній спільноті розробників та детальна документація спрощує зайомство з платформою	Високий, складна документація яка постійно доповнюється інформацією про нові функції та сервіси	Низький, знайомство з можливостями платформи відбувається у вигляді гри
Тарифні плани	Spark: безкоштовний, з квотами використання сервісів; Blaze: сплата лише за ті, послуги, які використовуються	Free tier: враховує квоти та є безкоштовним лише перші 12 місяців; Pay as you go: активується після року використання та сплата лише за ті, послуги, які використовуються	Springboard: безкоштовна але дуже обмежена; Cloud 9: для застосунків з невеликою аудиторією; Cloud 99: для великих підприємств

Згідно з головною метою роботи, необхідно визначити ту платформу яку буде доцільно запровадити у мобільний застосунок для пацієнтів з цукровим діабетом та їх лікарів за допомогою програмного каркасу Flutter. Згідно з документацією кожної платформи та пошуковому запиту у Dart Pub (менеджер пакетів для Dart та Flutter), існують необхідні плагіни та пакети для мобільних застосунків розроблених за допомогою Flutter, які є офіційними та створені спеціально для спрощення інтеграції API послуг платформи у проєкт. Тому у цьому питанні, усі рішення можна запровадити у проєкт.

Відповідно до таблиці 1.1 можна підвести підсумок:

а) Firebase є старішою серед оглянутих рішень та не втрачає популярність досі, доказом чого є наявність активній спільноті розробників, які діляться знаннями та досвідом на просторах Інтернету. Важливим чинником також служить те, що платформа була розроблена та постійно вдосконалюється Google, а значить вона є більш надійними BaaS серед інших.

б) найбільш привабливіша база даних у Backedless, тому що вона надає можливість за допомогою UI визначити логіку зв'язку між таблицями у БД, що можна порівняти з роботою у Microsoft Access. У той же час Firebase та AWS Amplify також цікаві з точки зору отримання досвіду роботи зі складними або невідомими БД.

в) процес вивчення можливостей Backedless є оригінальним та залучає багато користувачів до своєї платформи, а система винагород ще більш викликає зацікавленість до отримання нових знань. Firebase та AWS Amplify пропонують вичерпну документацію, однак розробники не будуть марнувати час на дослідження послуг, які вони не використовують у своїх проєктах.

г) тарифні плани розглянутих платформ дуже схожі й всі вони пропонують безкоштовний план для початківців. Тим не менш, вони мають принципові відмінності.

AWS Amplify безкоштовний лише перший рік користування й при умові що квоти, які були надані для початкового рівня не були перевищені. В

іншому випадку буде стягнуто плату з банківської карти, дані якої було надано при реєстрації.

З Firebase та Backendless все простіше: платформи не вимагають від користувачів банківських реквізитів одразу, а лише блокують доступ до сервісів у випадку перевищення трафіку. Але у цьому випадку виграє Firebase, бо квоти використання у платформи від Google значно вищі ніж у Backendless. Наприклад безкоштовний тариф сховища у Firebase охоплює 10 ГБ, у той час коли у Backendless це значення становить лише 1 ГБ [7]-[8].

Очевидно, що Firebase найбільш придатна для подальшого вивчення та впровадження у проєкт, бо є надійною та має більше безкоштовних ресурсів у порівнянні з іншими розглянутими платформами, що може бути достатньо для стартапу.

1.4 Короткий опис стану хворих на цукровий діабет та аналіз програмних рішень для них

За визначенням Всесвітньої організації охорони здоров'я, цукровий діабет (далі ЦД) – це хронічне захворювання, що розвивається в тих випадках, коли підшлункова залоза не виробляє достатню кількість інсуліну або коли організм не може ефективно використовувати вироблену їм інсулін – гормон, що регулює рівень вмісту цукру в крові. Загальним результатом неконтрольованого діабету є гіперглікемія, або підвищений рівень вмісту цукру в крові, що з часом призводить до серйозного пошкодження багатьох систем організму, особливо нервів і кровоносних судин [9]. небезпека цукрового діабету криється в численних швидкопрогресуючих судинних ускладненнях, в тому числі судин нервової системи (нейропатія), судин сітківки очного яблука (ретинопатія), судин нирок (нефропатія), коронарних судин і судин головного мозку [9].

З 1999 року за класифікацією, схваленої ВОЗ, виділяють:

а) ЦД 1 типу: застосовується до позначення групи захворювань, що розвиваються внаслідок прогресуючого руйнування бета-клітин підшлункової залози, що призводить до дефіциту синтезу проінсуліну та гіперглікемії, що вимагає замісної гормональної терапії

б) ЦД 2 типу: відноситься до захворювання, що розвивається у осіб з надмірним накопиченням жирової тканини, що мають інсулінорезистентність, внаслідок чого спостерігається надлишковий синтез проінсуліну, інсуліну та аміліну бета-клітинами підшлункової залози.

в) гестаційний ЦД, діагностований у II або III триместрі вагітності, що не був виявлений до вагітності.

г) інші специфічні типи ЦД:

1) моногенний тип успадкування: ЦД новонароджених і ЦД зрілого типу в людей молодого віку;

2) захворювання, при яких порушується екзокринна функція підшлункової залози: муковісцидоз, панкреатит;

3) застосування лікарських засобів або хімічних речовин (прийом глюкокортикоїдів, терапія ВІЛ/СНІД, після трансплантації органів).

За даними Міжнародної федерації діабету (IDF) від цієї хвороби та її ускладнень щороку гинуть мільйони людей, станом на 2019 рік цей показник становить 4,2 мільйони смертей. Загальні відомості про епідеміологічні особливості у світі [10]:

а) число хворих на діабет у віці 20-79 років на 2019 рік становить приблизно 463 млн осіб. Незважаючи на зниження швидкості приросту захворювання у деяких європейських країнах, щорічний приріст залишається на рівні 3,4% й до 2045 року цей показник зросте до 700 мільйонів.

б) 1 з 5 людей старше 65 років діабет, тобто 136 млн осіб.

в) понад 1,1 мільйона дітей та підлітків живуть з цукровим діабетом першого типу.

г) 374 мільйони людей мають підвищений ризик розвитку діабету другого типу.

Станом на листопад 2020 року в Україні зареєстровано 1 млн. 134 тис. людей, хворих на діабет, з них 181 тис. пацієнтів змушені постійно приймати інсулін. За даними КУ “Обласний інформаційно-аналітичний центр медичної статистики” ЗОР за 2019 рік в стаціонарах Запорізької області проліковано 5176 хворих з цукровим діабетом, в тому числі 407 дітей (0-17 років). Померло за рік 18 людей. За 9 місяців 2020 року проліковано 2307 хворих, в тому числі 241 дитина. Померло 22 людини [11].

Далі розглянемо програмні рішення, які використовуються хворими на цукровий діабет.

1.4.1 Аналіз існуючих програмних продуктів

Мобільний застосунок MySugr (див. рис 1.8) доступний для користувачів iOS і Android, надає змогу записувати рівень глюкози в крові протягом дня та, використовуючи ці значення, програма розраховує приблизний HbA1c через 7 днів.

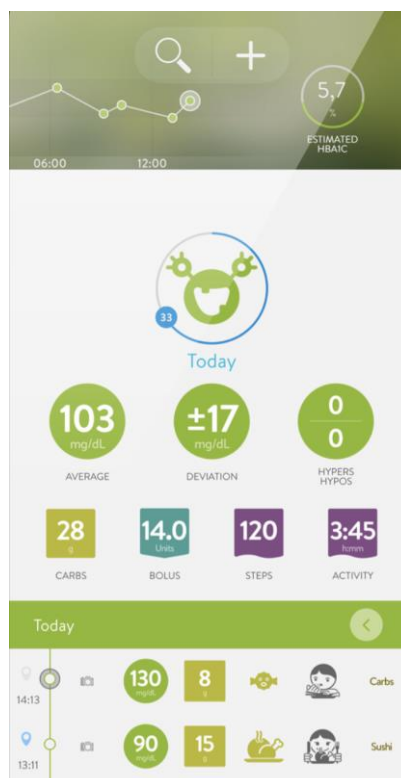


Рисунок 1.8 – Головний екран застосунку MySugr [12]

Застосунок також дозволяє записувати іншу інформацію, таку як споживання вуглеводів, маса тіла, фізична активність, артеріальний тиск, кетони, ліки та дози інсуліну. Щоб заохочувати позитивні звички (наприклад, запис дози інсуліну), застосунок використовує систему винагород, а також додаткові завдання. Застосунок простий у використанні та має хороші візуальні ефекти при перегляді даних. Також дозволяє швидко вводити дані або за допомогою глюкометра, який приєднано через Bluetooth, автоматично зберегти дані. Головним недоліком є те, що Pro версія має високу ціну, а безкоштовна – багато дратуючої реклами [12].

GlucoseTracker від Little Bytes (див. рис 1.9), який наявний лише для Android, надає функціональність щоденника самоконтролю з можливістю повідомлень про наступну ін'єкцію шляхом відправлення на електронну пошту.

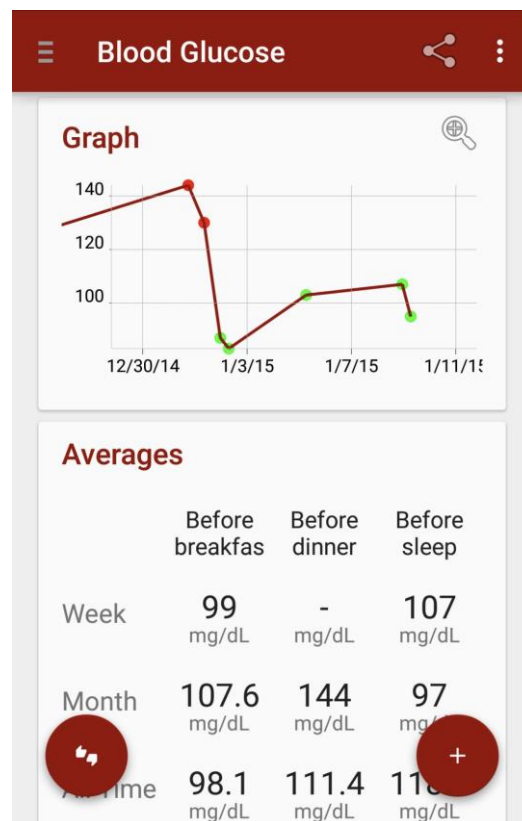


Рисунок 1.9 – Головний екран застосунку GlucoseTracker [13]

Програма дозволяє користувачеві записувати рівень глюкози в крові протягом дня та надає можливість встановити цільові діапазони глюкози в

крові для кожної події, наприклад перед сніданком, після обіду, перед сном. Також дозволяє записувати іншу інформацію, таку як ліки, артеріальний тиск, вага, HbA1c (%).

Застосунок простий у використанні: введення даних про кров здійснюється за допомогою повзунків, а не за принципом введення чисел напрямку. Незважаючи на те, що програма заснована в США, одиниці вимірювання рівня глюкози в крові, HbA1C та ваги тіла можна налаштувати відповідно до одиниць вимірювання які зручні користувачеві. Серед недоліків можна виділити те, що застосунок не безкоштовний та для можливості надсилати результати електронною поштою або отримувати доступ до всіх функцій, потрібно заплатити за програму [13].

Мобільний застосунок OneTouch Reveal (див. рис 1.10), який розроблений компанією LifeScan Europe та доступний для iOS та Android, має можливість внесення показників як вручну, так і за допомогою глюкометра.

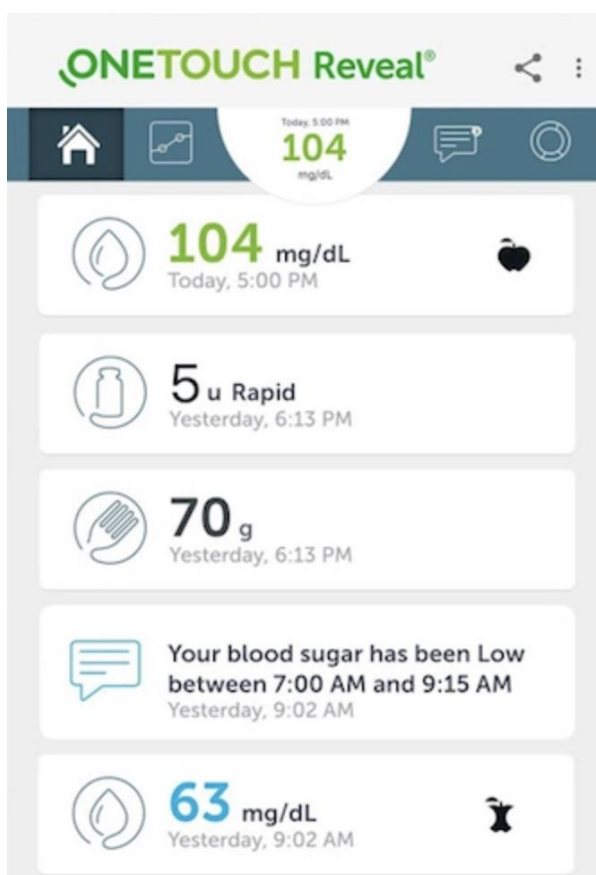


Рисунок 1.10 – Головний екран застосунку OneTouch [14]

Для підключення глюкометра необхідно придбати додатковий пристрій від компанії розробника програмного забезпечення. Застосунок також підтримує автоматичне введення даних про кількість вуглеводів, що вживаються, та веде спостереження про фізичну активність. Статистику про стан користувача можна переглянути у вигляді звітів та графіків. Програмний продукт надає можливість створення загального звіту та відправлення даних поштою та взаємодіє з глюкометром, відображує графіки статистичних даних користувача, має простий інтуїтивно-зрозумілий інтерфейс. Основні недоліки застосунку полягають у тому, що програмний продукт не надає рекомендацій дозування інсуліну та не завжди синхронізується з глюкометром [14].

Підводячи підсумок, можна провести порівняльний аналіз розглянутих існуючих рішень, які зручно представлені у вигляді таблиці 1.2.

Таблиця 1.2 – Порівняльний аналіз існуючих програмних продуктів

Показники	MySugr	GlucoseTracker	OneTouch Reveal
Зручність та простота інтерфейсу користувача	+	-	+
Можливість відображення графіків	-	+	+
Повідомлення про наступну ін'єкцію	+	+	-
Безкоштовна	-	-	-
Калькулятор інсуліну	+	-	+
Оффлайн режим	-	-	+
Наявність версій для різних ОС	+	-	+
Стабільність роботи	-	-	-

Серед проаналізованих програмних рішень взірцем для наслідування є застосунок OneTouch Reveal для, однак він має недоліки, серед яких те, що застосунок не надає рекомендацій дозування інсуліну та працює лише з глюкометрами від компанії розробника. Також, жоден з розглянутих рішень

не надає лікарям можливість отримувати дані своїх пацієнтів напряду через застосунок, лише шляхом отримання листа на електронну пошту.

1.5 Створення бізнес-моделі на основі отриманих даних

Для того щоб оцінити та отримати значення або представлення застосунку, що буде розроблено, у галузі медичних послуг, необхідно скласти бізнес-модель. Для цього буде використано підхід, який має назву “Канвас” (Canvas).

Бізнес-модель “Канвас” (або “Канва бізнес-моделі”) створили Александер Остервальдер та Ів Піньє. Ця схема на одну сторінку, яка описує усі бізнес-процеси компанії: пропозицію, інфраструктуру, споживачів, фінанси тощо [15].

Так як було розглянуто загальні характеристики існуючого стану хворих на діабет, а також проаналізовано існуючі програмні рішення для допомоги хворим на діабет, то можна створити канвас-таблицю на основі отриманих даних (див. табл. 1.3).

1. Споживчий сегмент. Ключовий елемент бізнес-моделі, який відповідає на запитання “Хто буде клієнтом?”. Виходячи з підсумків попереднього підрозділу, категорія яка може бути клієнтом майбутнього ПЗ – це хворі на діабет віком від 20 до 79 років. Але виходячи з того, імовірність того, що літні люди будуть використовувати застосунок, дуже мала, вікова категорія може становити від 20 до 50 років. Також користувачами будуть лікарські установи або лікарі, а також МОЗ.

2. Ціннісна пропозиція. Другий ключовий елемент бізнес-моделі. Це відповіді на наступні запитання: “Які цінності пропонуються споживачу?”, “Які проблеми програмний може вирішити?”, “Які потреби задовольняє?”. Планується, що застосунок буде зручним та простим у використанні, буде надавати змогу записувати рівень глюкози в крові протягом дня та вести статистику. Лікарі за допомогою системи будуть мати постійний доступ до

актуальних даних про пацієнтів. Більш того, система зможе надати статистичну інформацію про хворих на діабет у МОЗ, що дозволить міністерству отримати повну картину стану проблеми у країні та приблизно розрахувати бюджет на соціальні виплати хворим на діабет.

Таблиця 1.3 – Бізнес-модель “Канвас” майбутнього ПЗ

Ефективність			Цінність	
8. Ключові партнери Приватні лікарні та клініки; державне міністерство охорони здоров'я.	7. Ключові види діяльності Для пацієнтів та лікарів: комунікація та постійний доступ до інформації про стан пацієнта. Для держави: збір статистичних даних про стан проблеми в країні.	2. Ціннісна пропозиція Зручність та простота використання; лікарі мають постійний доступ до актуальних даних про пацієнтів.	4. Взаємовідносини з клієнтом Застосунок безкоштовний для пацієнтів, але є домовленості з приватними лікарнями або лікарями.	1. Споживчий сегмент Пацієнти з діагнозом цукровий діабет у віці 20-50 років. Лікарні та лікарі, яким необхідно знати про стан пацієнта онлайн.
	6. Ключові ресурси Команда розробників з 2-3 чол., ВааS, офісне приміщення, інвестиції лікарень або держбюджету.		3. Канали збуту Брошури у лікарнях; реклама та посилання на сайтах приватних клінік та на сайті МОЗ.	
9. Структура витрат Розробка та технічна підтримка; реклама; ВааS (сплата за використання ресурсів платформи).		5. Потоки надходження доходів Від приватних лікарень (лікарів) або державного бюджету, як застосунок який надає актуальні дані про пацієнтів.		

3. Канали збуту. Відповідає на запитання “Яким чином компанія доставляє продукт та інформацію про нього споживачам?”. Інформація про ПЗ може бути надана у лікарнях у вигляді реклами або як рекомендація від лікаря.

4. Взаємозв'язки з клієнтом. Планується що застосунок буде безкоштовним для пацієнтів, але будуть домовленості з приватними лікарнями або лікарями.

5. Потоки надходження доходів. Відповідає на запитання “За що клієнти дійсно готові платити?”. Наприклад, приватні лікарні, будуть готові сплачувати комісію, розмір якої буде залежати від кількості пацієнтів, які користуються системою. Також можна запросити інвестиції від держави, з метою фінансування системи, яка надає статистику про стан проблеми в країні.

6. Ключові ресурси або те, що необхідно для створення продукту. Основними та головними ресурсами можуть бути: команда розробників з 2-3 чол., бекенд як сервіс, сервіси для публікації застосунку (Play market та App store), офісне приміщення та інвестиції від лікарень або держбюджету.

7. Ключові види діяльності. Основна мета – вирішення проблем. Будь то лікарня, консалтингова чи туристична агенція тощо – клієнт звертається з проблемою, а застосунок повинен її вирішити. З цього боку, застосунок надає змогу непрямої комунікації між пацієнтом та лікарем та надає постійний доступ до інформації про стан пацієнта. Для держави збір статистичних даних про стан проблеми в країні.

8. Ключові партнери. Як було зазначено раніше, такими партнерами можуть стати приватні лікарні та клінік, а також державне міністерство охорони здоров'я.

9. Структура витрат. Виходячи, з ключових ресурсів, можна виділити основні витрати: це витрати на заробітну плату розробникам та тим хто відповідальний за технічну підтримку. В залежності від того які ресурси та квоти надає бекенд як сервіс, вони також потребують сплати. Також слід враховувати витрати на рекламу, до яких входить реклама на інтернет-ресурсах та в лікарнях.

Таким чином, програмне забезпечення, що розроблюється, буде унікальним, так як буде надавати лікарям онлайн доступ до стану здоров'я їх

пацієнтів. Такий підхід може покращити рівень обслуговування клієнтів у лікарнях.

На державному рівні, застосунок дозволить збирати статистичні дані про кількість хворих на діабет та їх стан, що надасть МОЗ повну картину про стан проблеми у країні та дозволить розробити стратегію і розрахувати бюджет, який розрахований на соціальну та фінансову допомогу громадянам, які хворі на діабет.

1.6 Висновок до першого розділу

У першому розділі було розглянуто провідні хмарні платформи: Firebase, AWS Amplify та Backendless. Проаналізовано переваги та недоліки кожної з них та було обрано Firebase для подальшого використання у майбутньому проєкті, так як це найдоступніша та надійна платформа з усіх представлених рішень. У наступному розділі необхідно розробити універсальну методологію, яка буде однаково ефективною для впровадження будь-якого BaaS у проєкт різної складності.

Також проаналізовано предметну область, проведено порівняння програм-аналогів для пацієнтів хворих на діабет, виявлено переваги і недоліки кожної з них. На основі досліджувань було створено бізнес-модель для того, щоб отримати базове уявлення про розроблюваний застосунок та перспективи його розвитку. Підводячи підсумок, розроблювана система має бути простою та зручною у використанні та задовольняти усім наведеним вимогам.

2 МАТЕРІАЛИ І МЕТОДИ

Для подальшої розробки програмного продукту необхідно вивчити яким чином працюють хмарні бази даних розглянутих платформ та визначити яка з них буде найкращою для впровадження у проєкт. Також важливим запитанням є те, яким чином краще спроектувати архітектуру проєкту, щоб вона була завжди готовою до переходу до нового постачальника серверних послуг, якщо це буде потрібно.

2.1 Методи роботи систем управління базами даних

У 2000 році Ерік Брюер, професор комп'ютерних наук, висунув гіпотезу, суть якої можна описати так: у розподіленій системі неможливо забезпечити одночасне виконання всіх трьох умов: узгодженості, доступності та стійкості до поділу або скорочено CAP (Consistency – Availability – Partition tolerance). Теорема CAP стверджує, що розподілена система може мати не більше двох властивостей із трьох одночасно [16].

Теорема CAP складається з трьох основних властивостей (див. рис. 2.1), оскільки вони стосуються розподілених сховищ даних [17]:

- а) узгодженість. Усі зчитування отримують останній запис або помилку.
- б) доступність. Усі зчитування містять дані, але вони можуть бути не релевантними.
- в) стійкість до поділу. Система продовжує працювати, незважаючи на збої в мережі.

У звичайній роботі сховище даних може забезпечувати всі три функції. Але теорема CAP стверджує, що коли розподілена база даних зазнає збою в мережі, то можна забезпечити узгодженість або доступність. В інший час можна надати всі три. Але в разі збою в мережі необхідно зробити вибір. У теоремі допуск на розподіл є обов'язковим. Передбачається, що система

працює з розподіленим сховищем даних, тому система за своєю природою працює з мережевими розділами. Збої в мережі відбуватимуться, тому для надання будь-якого надійного сервісу необхідна стійкість до розділу. Коли відбувається збій мережі, можна вибрати; гарантувати узгодженість або доступність. Тобто висока узгодженість забезпечується ціною меншої доступності, а висока доступність забезпечується ціною меншої узгодженості [16].

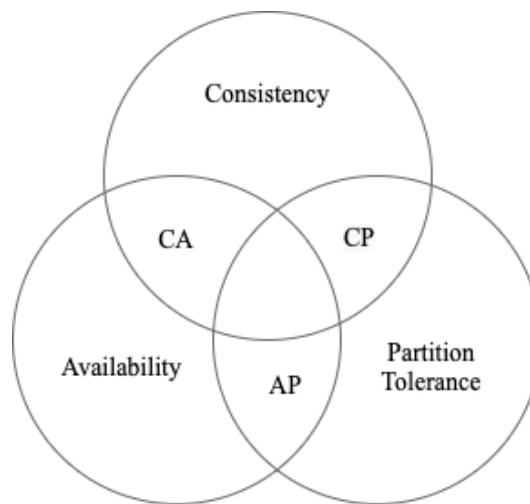


Рисунок 2.1 – Схематичне представлення теореми CAP

Наприклад, користувач робить запит до бази даних, а мережева база даних має повернути значення. То яке значення повертається з бази даних, залежить від вибору: узгодженість або доступність. На запит можна відповісти користувачеві з поточним значенням на сервері, пропонуючи високодоступну послугу. У цьому випадку, немає гарантії, що значення є останнім, яке було подано до бази даних. У випадку коли треба гарантувати високу узгодженість, то необхідно дочекатися нового запису або повернути запит про помилку [17].

Вибір узгодженості та доступності відбувається під час вибору типу бази даних. Бази даних NoSQL можна класифікувати на основі того, чи підтримують вони високу доступність чи високу узгодженість. З цього боку, бази даних розглянутих BaaS, згідно офіційним джерелам, мають наступні властивості:

а) бази даних Firebase – NoSQL, орієнтовано на узгодженість, іншими словами якщо ця база даних зазнає збою мережі, то вона орієнтована на узгодженість ніж на доступність.

б) AWS Amplify пропонує безліч систем управління даними, у тому числі власного виробництва – Amazon Aurora, яка є реляційною БД та орієнтована на доступність. Також платформа може працювати з іншими БД.

в) Backendless пропонує масштабовану та динамічну базу даних яка надає можливість вибору та між доступністю та узгодженістю.

Вимоги CAP теорема застосовуються до розподілених систем загалом, а не лише до баз даних. Однак на сьогодні немає потреби в такому категоричному поділі систем на класи CA/CP/AP, як це передбачає CAP теорема. Більше того, така класифікація навіть вважається помилковою, оскільки широка інтерпретація CAP не враховує потреб сучасних розподілених баз даних і застосунків Big Data, а тому її застосування до цих систем не зовсім правильно. Тому, за останні 20 років застосування теорема CAP довела свою спроможність, проте видалася недостатньою. Так у 2010 році з'явилася PACELC як розширення CAP. Теорема PACELC, описана Деніелем Дж. Абаді, та свідчить, що у разі поділу мережі у розподіленій комп'ютерній системі необхідно вибирати між доступністю та узгодженістю, але навіть якщо система працює нормально у відсутності поділу, потрібно вибирати між затримками та узгодженістю. На рисунку 2.2 представлено схематичне уявлення PACELC теорема[17].

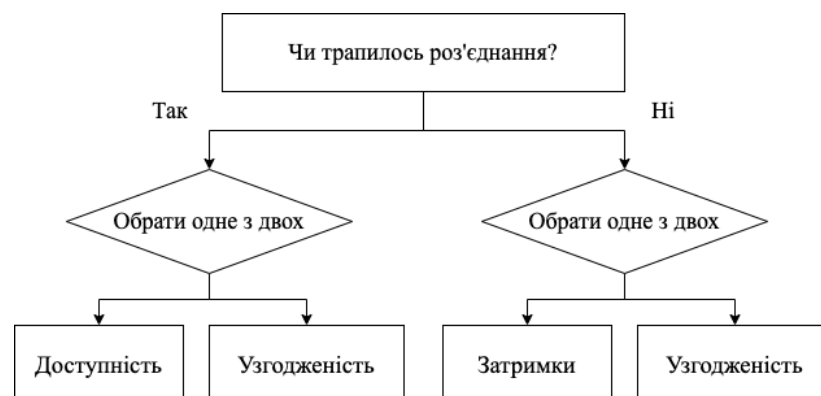


Рисунок 2.2 – Схематичне представлення теорема PACELC

Таким чином, PACELC розширює та уточнює CAP теорему, регулюючи необхідність пошуку компромісу між затримкою часу та узгодженістю даних у розподілених системах великих даних.

Підводячи підсумок, незважаючи на те, що розробники мобільного застосунку лише використовують наявні рішення BaaS, а не розробляють самостійно, їм необхідно знати як працюють наявні БД та, в залежності від вимог до майбутнього програмного продукту, обрати належний сервіс з надання серверних послуг. З точки зору того, яка база даних буде оптимальною для майбутнього застосунку, а саме ПЗ для пацієнтів хворих на діабет, то важливим чинником є орієнтованість на узгодженість, бо стан статистичних даних повинен бути завжди релевантним. Отже усі розглянуті платформи та їх БД можуть бути використані у майбутньому застосунку.

2.2 Методи впровадження хмарних сервісів у програмне забезпечення

Стосовно розробки застосунку, то на даний момент розроблено безліч архітектур, які використовуються для розробки ПЗ, у тому числі для впровадження стороннього фреймворку у проєкт. Звичайно, команда або окремий розробник обирає саме тій спосіб, який зручніший. Але, у той же час не можна забувати про те, що неможливо передбачити майбутнє. Тому треба використати ту архітектуру у проєкті, яка буде гнучкою та завжди готовою до змін.

С початку розвитку ІТ індустрії, було скоєно багато помилок у розробці ПЗ, що призвело до невдач та повному припиненню існування програмних рішень, розробники яких не змогли впоратися з постійними змінами, що зробило такі ПЗ неконкурентоспроможними. Тому, методом проб та помилок, було запроваджено загальні принципи до архітектури ПЗ, які актуальні досі [18]:

а) незалежність від фреймворку. Архітектура залежить від існування будь-якої бібліотеки. Це дозволяє використовувати фреймворк як інструмент замість того, щоб втискати свою систему в рамки його обмежень.

б) тестованість. Бізнес-правила можуть бути протестовані без інтерфейсу користувача, бази даних, веб-сервера або будь-якого іншого зовнішнього компонента.

в) незалежність від UI. Інтерфейс користувача можна легко змінити, не змінюючи решту системи. Наприклад, веб-інтерфейс може бути замінений на консольний без зміни бізнес-правил.

г) незалежність від бази даних. Можна змінити Oracle або SQL Server на MongoDB, BigTable, CouchDB тощо. Бізнес-правила проєкту не пов'язані з базою даних.

д) незалежність від будь-якого зовнішнього сервісу. За фактом, бізнес-правила проєкту нічого не знають про зовнішній світ.

Дотримання цих принципів може буди здійснено за допомогою впровадження чистої архітектури, яку запропонував Роберт Мартін (рис. 2.3). Класично схема чистої архітектури виглядає як кільця цибулі в розрізі, але протягом її існування, серед розробників було багато суперечок, щодо того, як правильно її розшифровувати. Тому саме таке уявлення даної схеми, яку надано далі, більш зрозуміліше для отримання уявлення чистої архітектури.

Така схема складається з наступних компонентів:

а) Entities або сутності це бізнес-логіка загальна для багатьох застосунків та визначаються бізнес-правилами підприємства. Сутність може бути об'єктом з методами або вона може бути набором структур даних і функцій. Не має значення, як довго сутність може бути використана в різних застосунках. В одиночних застосунках сутністю є бізнес-об'єкти цієї програми. Вони інкапсулюють найзагальніші високорівневі правила. Найменш ймовірно, що вони зміняться за будь-яких зовнішніх змін. Наприклад, вони не повинні бути порушені при зміні навігації сторінок або правил безпеки. Зовнішні зміни повинні впливати на шар сутностей.

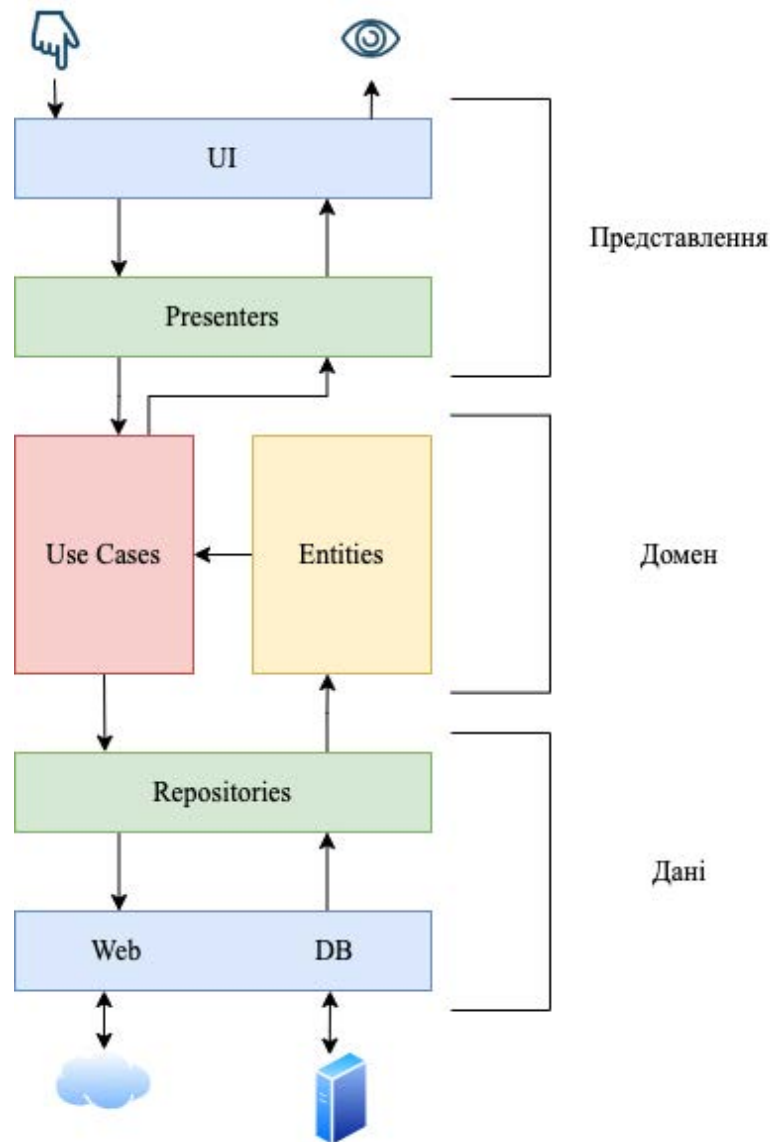


Рисунок 2.3 – Схематичне представлення шарів чистої архітектури [18, с.204]

б) Use Cases або сценарій використання. У цьому шарі реалізується специфіка бізнес-правил. Він інкапсулює та реалізує всі випадки використання системи. Ці сценарії реалізують потік даних і з шару Сутностей для реалізації бізнес-правил. Зміни в цьому шарі не повинні впливати на Сутність, а також цей шар не може бути порушений зовнішніми змінами (бази даних, інтерфейс користувача або фреймворк).

в) Presenters та Repositories – адаптери між Use Cases та зовнішнім світом. Програмне забезпечення в цьому шарі є набором адаптерів, які перетворюють дані з формату найбільш зручним для Сценаріїв і Сутностей, у формат найбільш зручний для подальшого використання, наприклад в БД.

Часто моделі це структури даних, які передаються від контролерів до сценаріїв, а потім назад зі сценаріїв до UI. Так само, дані перетворюються, у цьому шарі, з форми найбільш зручним для Сценаріїв і Сутностей, у форму, найбільш зручною для постійного зберігання, наприклад, у БД.

г) UI, Web та DB – фреймворки, БД, інтерфейс користувача тощо. Як правило, у цьому шарі не пишеться багато коду, за винятком коду, який спілкується із внутрішніми колами.

Основним правилом, що робить цю архітектуру робочою є Правило Залежностей. Як вже було зазначено, класично схема чистої архітектури виглядає як кільця цибулі, де Entities у середні цієї схеми, далі Use Cases, потім адаптери і останній шар – фреймворки. Згідно з Правилем Залежностей, залежності у вихідному колі можуть вказувати тільки усередину. Ніщо з внутрішнього кола не може знати щось про зовнішнє коло та не може вказувати на зовнішнє коло. Це стосується функцій, класів, змінних тощо. Більше того, структури даних, що використовуються у зовнішньому колі, не повинні бути використані у внутрішньому колі, особливо якщо ці структури генеруються фреймворком у зовнішньому колі [18, с.204].

При дотриманні принципів чистої архітектури будь-який застосунок, який було створено з дотриманням цих принципів, буде надавати змогу швидко змінювати логіку роботи з сервером, а також бути модульним для тестування. Саме тому частина, яка відповідальна за роботу з веб та БД повинна залежати від абстрактного класу або інтерфейсу.

Кожен з розглянутих сервісів, які було розглянуто у першому розділі, має API для Flutter фреймворку. Але жоден з постачальників не описує як правильно впровадити API у проєкт, а лише описує що роблять методи API [19]-[21]. Тому головним завданням є опис застосування принципів чистої архітектури у проєкті, який використовує BaaS.

Надана схема (рис. 2.3) дуже абстрактна, та не надає повного уявлення про архітектуру проєкту. Тому, на рисунку 2.4 було більш детальніше

проілюстровано компоненти чистої архітектури. Така схема допомагає повністю уявити роботу проєкту та місце окремих компонентів у системі.

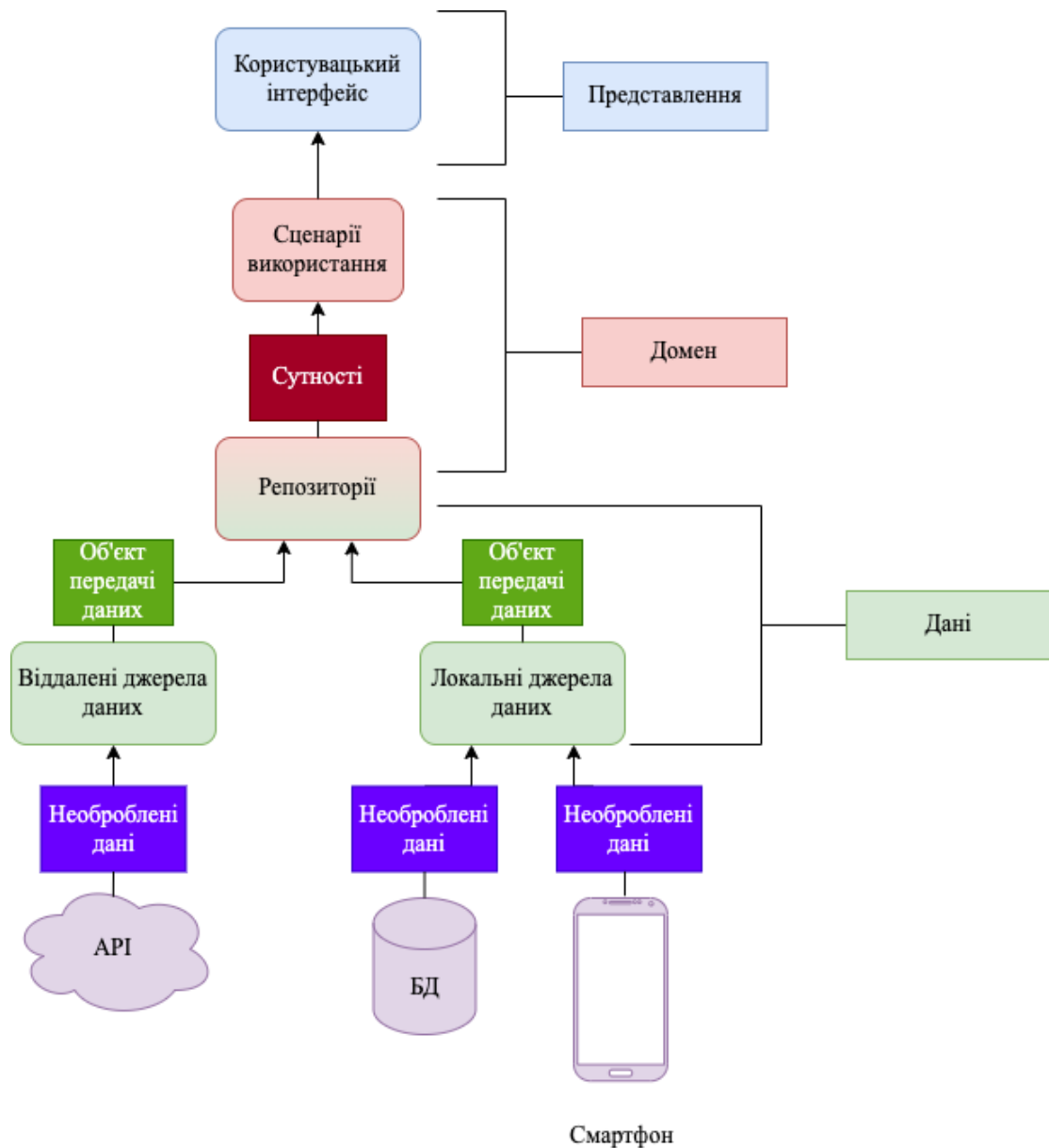


Рисунок 2.4 – Представлення архітектури проєкту на основі загальної схеми

Також основною проблемою у процесі розробки може бути те, що на практиці дуже часто DTO (об'єкт передачі даних) та Entities (сутності) поєднують у один клас, що може стати проблемою у майбутньому. А саме після того, як будь-які поля у відповіді від бекенда буде змінено, потрібно змінювати код скрізь, де є згадка про них. Це може призвести до того що у проєкті з'являться багато багів які буде важко або неможливо виправити. Тому

дуже важливо, щоб сутності було створено згідно з потребами застосунку, а не на вимогу запитів API. Сутність повинна не залежати від серверної частини та необхідно щоб саме вона визначала бізнес-логіку програмного продукту.

Далі джерела даних, або їх можна описати як API, які надають змогу спілкування застосунку з сервером або з локальною БД. Реалізація цього шару повинна бути таким чином, щоб в жодному з компонентів системи не було жодного згадування про нього. Використовуючи такий принцип, у майбутньому буде легко та швидко змінити постачальника бази даних без втручання в код, який відповідальний за інтерфейс користувача, іншими словами гарантує відмовостійкість (Fault Tolerance). На прикладі програми, що буде розроблена, тут потрібно описати що роблять методи API, інформацію про які надано у офіційних джерелах постачальників серверних послуг [19]-[21].

2.3 Висновок до другого розділу

У другому розділі було розглянуто теореми, які використовуються у розробці розподілених систем, та вивчено якими властивостями володіють розглянуті БД. Також було описано загальні принципи чистої архітектури та було виявлено, що постачальники BaaS не надають інформації щодо грамотного впровадження API у проєкт. Тому було запропоновано, за допомогою принципів чистої архітектури, впровадити API бекенд як сервісу у проєкт. Запропоновані архітектурні рішення надають змогу, у випадку необхідності, замінити постачальника бекенд послуг, витратити мінімум часу та зусиль на реалізацію логіки з новим API. Передбачається, що у наступних розділах, на прикладі роботи з Firebase, буде впроваджено розглянуті принципи та архітектурні особливості.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

Метою розробки є створення застосунку, який використовує бекенд як сервіс та хмарну базу даних. Застосунок буде розроблено за допомогою фреймворку Flutter, функціонувати під керуванням ОС Android або iOS та взаємодіяти з сервісом який надає послуги бекенд-сервісу (BaaS). Другорядним завданням є створення програми, що повинна ввести звіт про стан пацієнта хворого на цукровий діабет та зберігати ці дані у БД, до якої будуть мати доступ лікарі та лікарні, які обслуговують пацієнта. Застосунок повинен бути простим, зручним та інтуїтивно зрозумілим.

3.1 Бізнес-логіка програмного продукту

Принцип роботи мобільного застосунку, що буде розроблено, доволі простий. Користувач реєструється або авторизується у системі як пацієнт або лікар. Якщо це пацієнт, він зможе:

- а) змінити дані профілю: ім'я пацієнта та електрону пошту лікуючого лікаря;
- б) зберегти нові показники крові за допомогою форми вводу даних;
- в) переглянути статистику;
- г) поділитися особистим ідентифікатором з лікарем.

Також буде додано функціональність створення облікового запису для лікаря, для того, щоб він мав змогу спостерігати за станом своїх пацієнтів. Тому треба додати такі можливості:

- а) додавати клієнтів до списку пацієнтів;
- б) переглядати список пацієнтів та мати доступ до статистичних даних про стан їх здоров'я.

На рисунку 3.1 зображено загальне схематичне представлення роботи такої системи. Як видно на схемі, пацієнти та лікарі мають спільний доступ до бази даних. Передбачається, що для отримання лікарем доступу до даних

пацієнта, пацієнту буде необхідно поділитися власним унікальним ідентифікатором з лікарем.



Рисунок 3.1 – Загальна схема взаємодії користувачів з застосунком

Для більш детального уявлення про роботу застосунку з різними типами користувачів було розроблено діаграму прецедентів для застосунку, що розробляється (див. рис. 3.2).

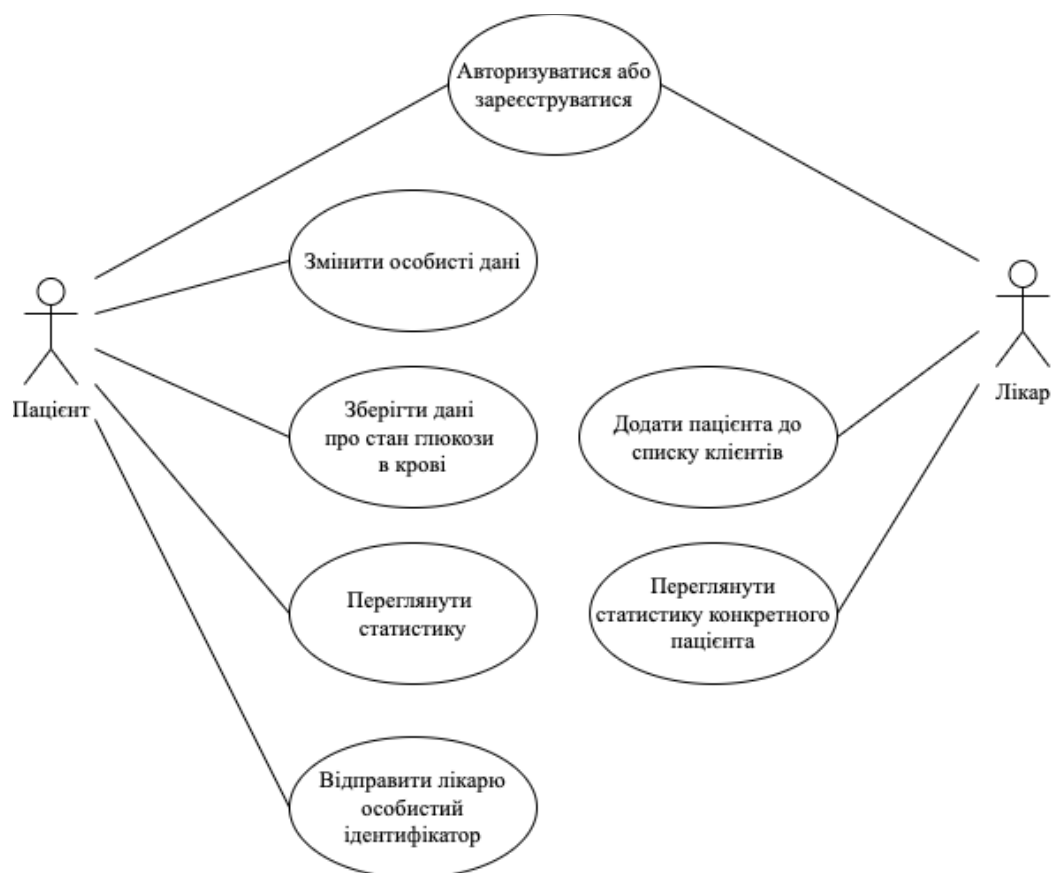


Рисунок 3.2 – Діаграма прецедентів

Як можна побачити, логіка роботи застосунку проста та не має

складних сценаріїв, тому основну увагу можна приділити реалізації програмного продукту, іншими словами, яким чином розробити застосунок.

3.2 Архітектура програмного продукту

Ґрунтуючись на принципах архітектури, які було запропоновано у другому розділі, для розробки пропонується адаптивний архітектурний шаблон похідний від MVP (див рис. 3.3), де класи з суфіксом Page виконують функцію інтерфейсу користувача, з суфіксом Repo – інтерфейс та реалізацію логіки взаємодії зі сервісами постачальника даних, а також з суфіксом Presenter, який поєднує у собі утримувача логіки представлення та сценарій використання, або, іншими словами, є шаром між Page та Repo.

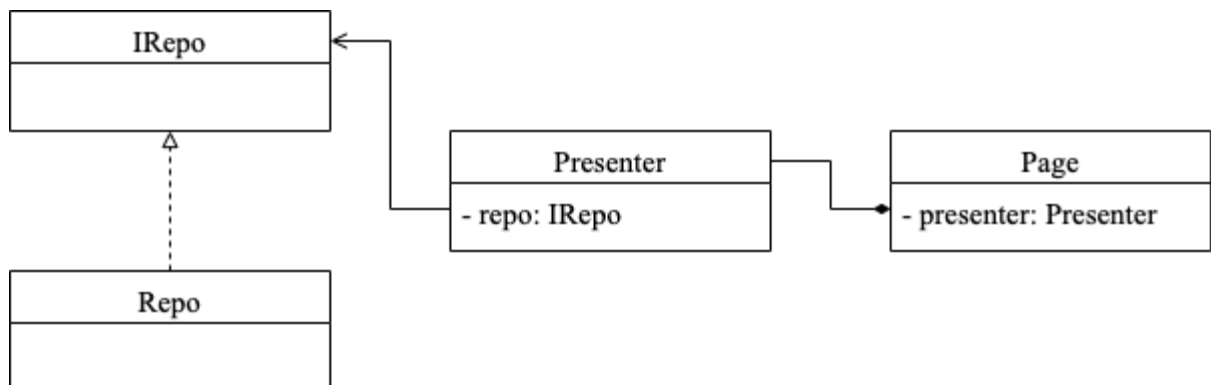


Рисунок 3.3 – Спрощена діаграма класів запропонованого адаптивного архітектурного шаблону

Repo-інтерфейси (IRepo) та похідні від них класи (Repo) з'єднують застосунок та функціональність серверного постачальника послуг, а саме реалізують методи роботи з сервером. Система повинна бути модульною, для того щоб при зміні API, не потрібно було втручатися в код який залежний від UI. Саме тому було змодельовано абстрактний шар репозиторію про який знає Presenter, а реалізація репозиторію буде ініційована на початку роботи програми. Таким чином, за потребою, можна легко імплементувати та без

небажаного втручання у код та змін у проєкті, впровадити нову реалізацію репозиторію.

Слід також звернути увагу, на необхідність створення об'єктів передачі даних для роботи з сервером. Часто розробники ігнорують створення так званих DTO, та реалізують логіку цих класів у класах сутностей, що може спровокувати проблеми у подальшому процесі розробки. Наприклад, після того як запити та моделі на сервері будуть змінені, виникне необхідність змінювати логіку усього проєкту. Тому дуже важливо, щоб сутності у проєкті були незалежними від серверної частини та визначали бізнес-логіку програмного продукту, а про об'єкти передачі даних було відомо тільки реалізаціям репозиторіїв. Схематично це може виглядати так, як показано на рисунку 3.4.

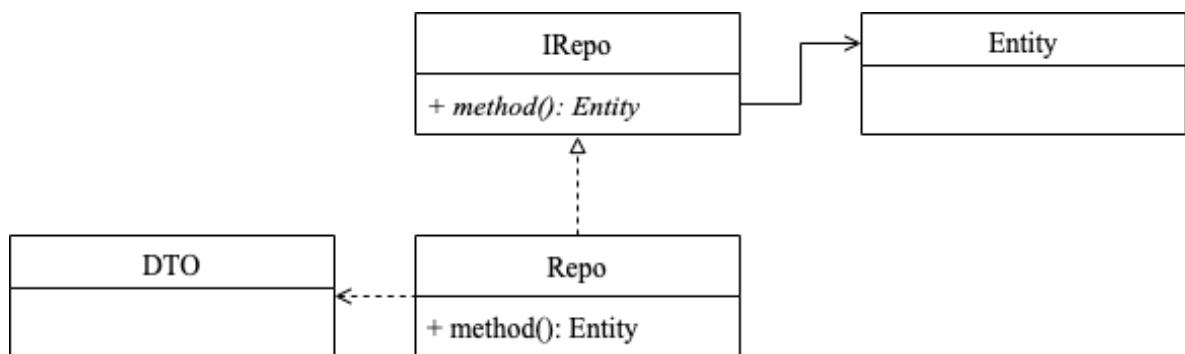


Рисунок 3.4 – Спрощена діаграма класів, яка демонструє зв'язок сутності та об'єкту передачі даних

Синтаксичний аналіз або парсінг можна реалізувати двома шляхами, де перший перетворювати на сутності напрямку у методі-реалізації, у випадку якщо відповідь від серверу досить проста, або створити об'єкти передачі даних де буде здійснюватися створення об'єкту класу з отриманої відповіді від бекенду.

Використовуючи запропонований підхід, може здаватися, що це марнування ресурсів, але у довгостроковій перспективі така стратегія заощадить час, гроші та нервові клітини розробників, що позитивно

відобразиться на застосунку, його архітектура буде знаходитися в стані, при якому проєкт буде завжди готовий до змін або до реалізації нової функціональності, які передбачені вимогами до проєкту.

Передбачається, що застосунок буде мати два типи користувачів: пацієнт та лікар. Тому сутності будуть мати властивості наведені на рисунку 3.5. Як видно з діаграми класів, Patient і Doctor успадковуються від User, який має спільні властивості: id, email, fullName. Patient має додаткове поле – docEmail – значення якого може бути null, якщо користувач не додасть електрону пошту лікуючого лікаря за допомогою застосунку. Сутність Doctor теж має додаткове поле patientIds, що визначає перелік ідентифікаторів пацієнтів, стан яких відстежує лікар.

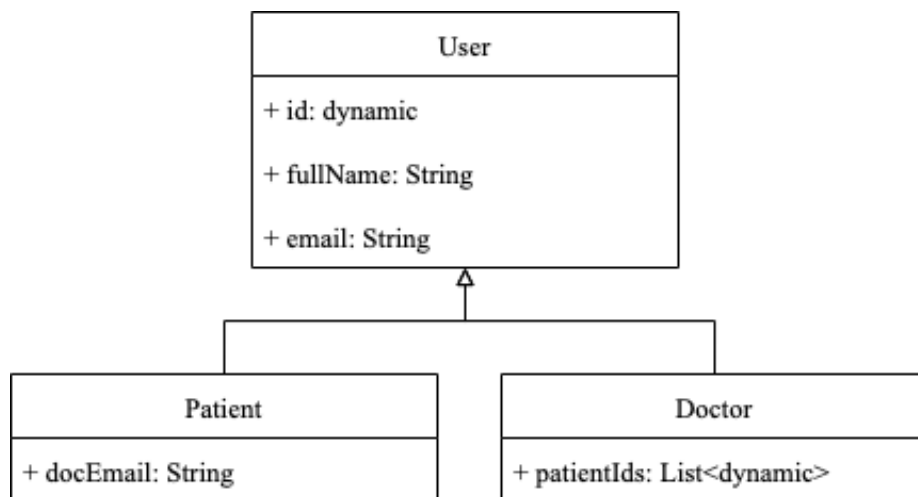


Рисунок 3.5 – Діаграма класів сутності User

Ще однією з важливих сутностей є та, що відповідає за статистику. Однак її представлено абстрактним класом **Statistic**, тому що, у майбутньому застосунок може обробляти не тільки дані глюкози у крові пацієнта, а й інші показники. Так як загальною властивістю для усіх одиниць статистики є час зняття показників, тому клас **Statistic** має властивість `DateTime` та метод `compareTo(Statistic other)` для сортування дочірніх класів за датою у списках (див. рис. 3.6).

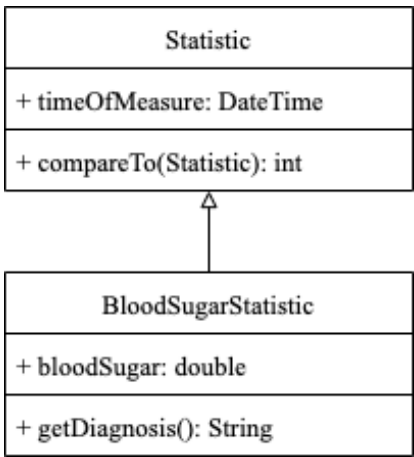


Рисунок 3.6 – Діаграма класів сутності Statistic

У процесі розробки постачальник сервісних послуг може змінитися на будь-який інший. Тому, з урахуванням вимог до проєкту, пропонується реалізувати патерн проєктування “Фабрика”, який в залежності від платформи та типу користувача створить відповідну реалізацію логіки. Схема такої структури виглядає так, як показано на рисунку 3.7.

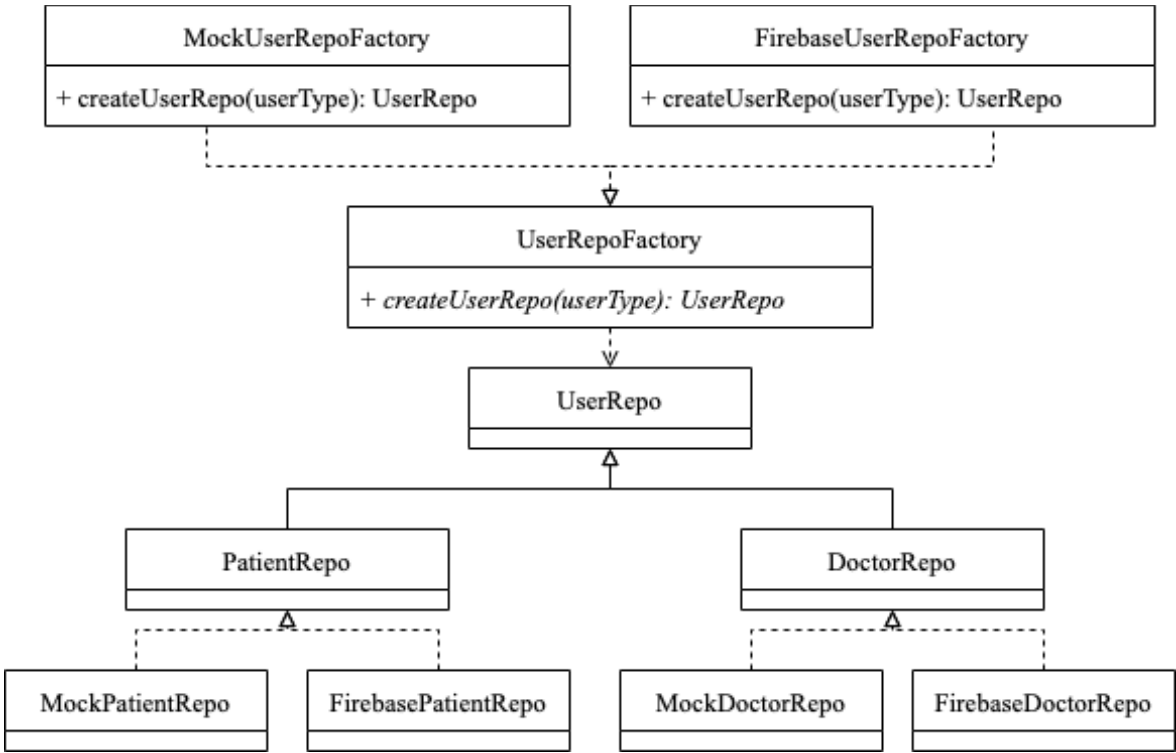


Рисунок 3.7 – Діаграма класів патерну “Фабрика”

Так як один застосунок повинен виконувати різні функції, які обумовлені типом користувача, важливо поділити логіку на ту що буде виконуватися пацієнтом та на ту що лікарем. Звісно вони можуть мати спільні риси взаємодії зі застосунком, тому PatientRepo та DoctorRepo успадковуються від UserRepo, який має загальні методи для дочірніх класів.

Використовуючи наведений принцип організації класів, можна спростити створення класів з логікою для пацієнтів та лікарів та забезпечити безпечність даних користувачів, тобто, наприклад, користувач з типом пацієнт, не зможе використати логіку призначену лікарю, та навпаки.

Наступним постає питання: яким чином фабрика дізнається коли слід створити відповідний репозиторій? Пропонується, що репозиторій відповідальний за авторизацію вирішить цю проблему (див. рис. 3.8).

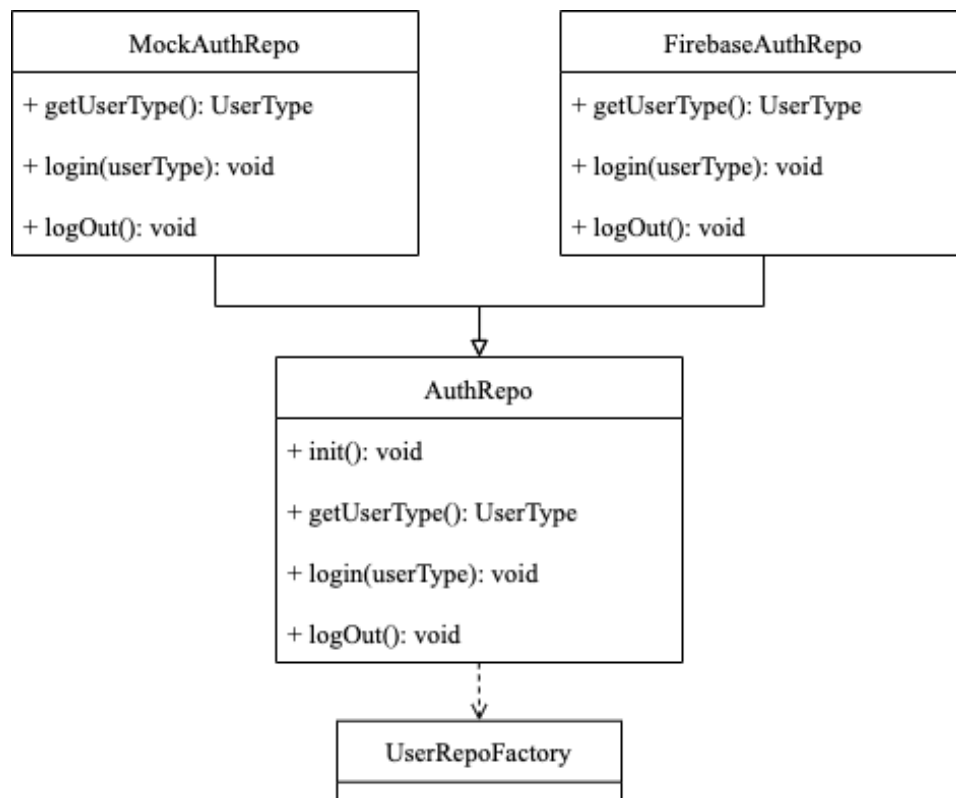


Рисунок 3.8 – Діаграма взаємодії класів авторизації та фабричного класу

В даному випадку клас **AuthRepo** не інтерфейс а повноцінний клас, кожен метод якого знає яким чином та при яких умовах використати фабрику

UserRepoFactory для створення відповідного репозиторію. Класам, які успадковуються від AuthRepo, треба лише додати логіку взаємодії з сервером та визвати метод батька відповідно до методу який реалізується.

Важливо відзначити, що не бекенд визначає логіку застосунку. Якщо застосунок не може хоча б імітувати роботу з сервером, то архітектуру такого проєкту слід реорганізувати, таким чином, щоб заміна логіки реального API на фіктивний здійснювалося в одному місці проєкту. Застосунок і бекенд це дві незалежні одиниці, які звісно необхідні один одному, однак це не значить що вони залежні один від одного.

3.3 Висновок до третього розділу

У даному розділі було розглянуто бізнес-логіку майбутнього застосунку. Так як вона виявилася простою, особливу увагу було звернено на реалізацію програмного продукту.

На основі архітектурних принципів, було розроблено адаптивний архітектурний шаблон. Основну увагу приділено, необхідності створення об'єктів передачі даних для роботи з сервером та розділення шару логіки між типами користувачів. Описано головні сутності проєкту, які визначають бізнес-логіку застосунку. На основі патерна “Фабрика” було розроблено архітектуру для створення репозиторіїв в залежності типу зареєстрованого користувача.

4 РОЗРОБКА ПРОГРАМИ

У поточному розділі буде доведено чому програмний каркас Flutter обрано як основний інструмент розробки. Для цього буде проаналізовано інші програмні каркаси та середовища розробки, які можуть бути використані для розробки та які зможуть задовольнити потреби майбутнього програмного продукту. Також буде описано процес розробки, який буде ґрунтуватиметься на архітектурному шаблоні, який було створено у попередньому розділі. Слід зауважити, що основною метою є створення застосунку, який використовує бекенд як сервіс та хмарну базу даних, тому більше уваги буде приділятися впровадженню BaaS у проєкт.

4.1 Обґрунтування вибору програмного каркасу для розробки програми

У 2021 році вибір серед Flutter, або ReactNative, або Xamarin постає гострим питанням, яке постає перед кожним розробником, компанією або власником застосунку.

Одним з найважливіших чинників, який слід враховувати при виборі кросплатформної системи розробки – це ефективність застосунків. ReactNative та Xamarin пропонують нативну розробку, але код Dart, який використовує програмний каркас Flutter, компілюється у бібліотеку C/C++, що значить він близький до нативного коду. Це покращує швидкість взаємодії з застосунком та забезпечує кращу продуктивність [22].

Apple iOS SDK було випущено у 2008 році, а Google Android SDK – у 2009 році. Ці два SDK були засновані на різних мовах: Objective-C та Java. Однак зараз платформи еволюціонували та використовують Swift та Kotlin відповідно до платформи.

Тим не менш, вони використовують однаковий принцип роботи застосунків як зображено на рисунку 4.1 [23].

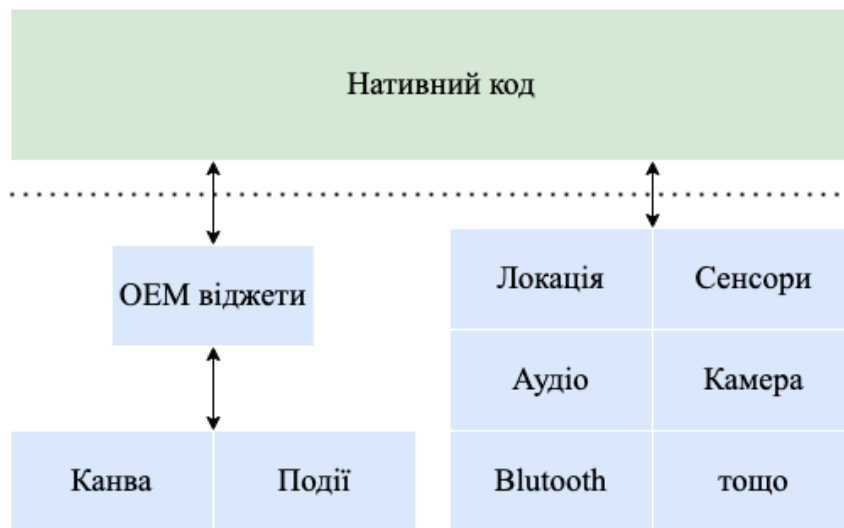


Рисунок 4.1 – Схематичне зображення відносин застосунку з платформою

ReactNative та Xamarin для відображення компонентів системи використовують міст (bridge), що робить розробку залежною від платформ. Іншими словами, для створення нестандартного компонента необхідно писати код для обох платформ. Принцип такої роботи наведено на рисунку 4.2 [24].

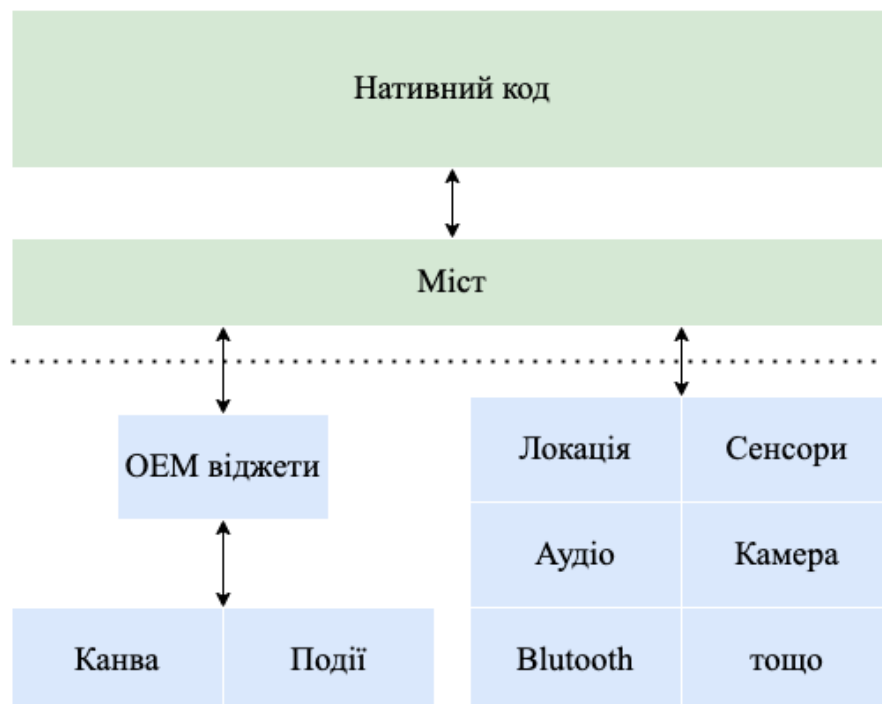


Рисунок 4.2 – Схематичне зображення роботи ReactNative з платформою

Передумовою виникнення Flutter, було те що в 2014 році, щоб досягти

значної швидкості, команда Google Chrome експериментувала з відображенням вмісту сторінок. Метою експерименту була перевірка: чи можна прискорити вимальовування графічних елементів, якщо відмовитися від традиційної моделі верстки, водночас позбувшись від зворотної сумісності.

Основою для експериментів став двигун Blink, а результатом – приріст продуктивності в 20 разів за рахунок таких змін [25]:

- а) впровадження простого набору обмежень, що впливають на стан елемента на екрані: мінімальна/максимальна ширина і висота;
- б) кожен компонент визначає свою просту верстку-модель (центрування, падінг, колонка).

Мала кількість правил дозволяє оптимізувати верстку на рівні компонента. А проста модель, що розповсюджується від батька вниз по дереву, дозволила домогтися лінійної залежності між кількістю елементів в дереві і часом створення верстки. Все відбувається настільки швидко, що двигун Blink не поділяє “створення” і “модифікацію”. Замість цього, у разі змін батьківського віджета, він знову створює все піддерево. Результат цього експерименту було взято за основу при розробці Flutter.

Особливістю архітектури програмного каркасу Flutter є те, що він сам малює кожен піксель на канві, контролює жести і анімацію. Фреймворк не вживає мости та разом з тим OEM-віджети, як це робить ReactNative. Замість цього, команда, що розробила Flutter, створила два набору віджетів для основних мобільних платформ: `StatefullWidget` та `StatelessWidget`. Вони відповідальні за створення UI-компонентів для обох мобільних платформ, та повторюють поведінку, заснованої на досвіді користувача. Безпосередньо з мобільною платформою (геолокація, звук, bluetooth) взаємодія відбувається через Platform Channels (див рис. 4.3). Це мабуть єдиний значний мінус Flutter – програмний каркас надає змогу намалювати все що завгодно, але нативні можливості реалізуються тільки за допомогою Platform Channels [26].

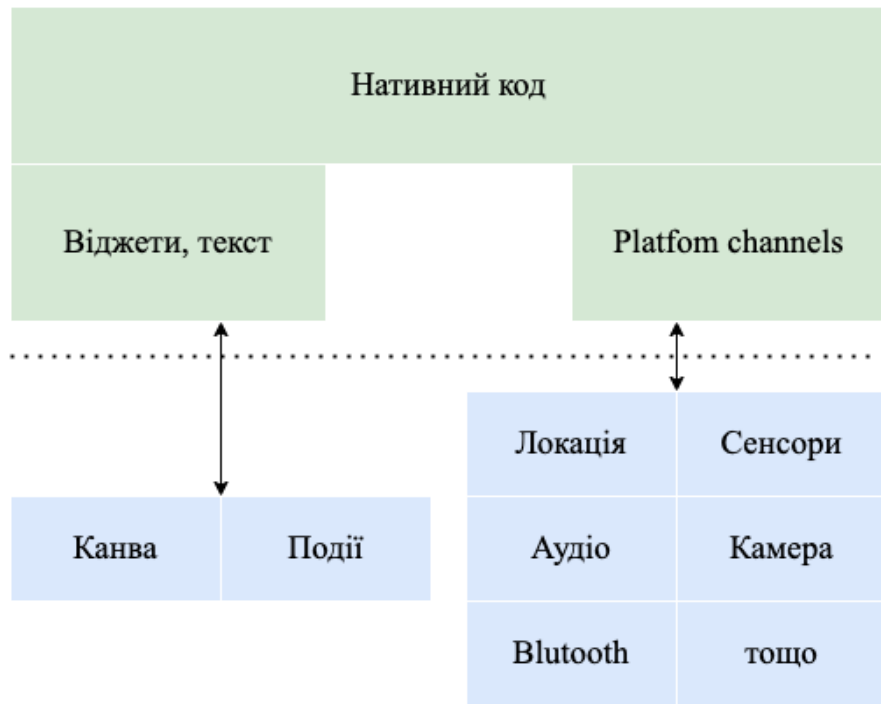


Рисунок 4.3 – Схематичне зображення роботи застосунку написаному на Flutter

Для кращого розуміння відмінностей наявних програмних каркасів, нижче наведено таблицю 4.1 із їх порівняльним аналізом.

Таблиця 4.1 – Порівняльна характеристика фреймворків [27]

Критерій	Flutter	ReactNative	Xamarin
Дата випуску	2018	2015	2011
Розроблена	Google	Facebook	Microsoft
Мова представлень	Dart	JSX	XAML або xamarin.forms
Мова програмування	Dart	JavaScript	C#
Залежність від нативних платформ	низька	висока	дуже висока
Ліцензія	BSD	MIT License	MIT License

Виходячи з отриманих даних таблиці 4.1, для розробки застосунку обрано Flutter, так як це новий програмний каркас, який має більше переваг серед інших фреймворків таких як React Native та Xamarin.

4.2 Особливості та вимоги для застосунку створеному за допомогою програмного каркасу Flutter

Так як планується, що застосунок буде розроблений за допомогою програмного каркасу Flutter, слід розглянути основні архітектурні особливості та вимоги до розробки програмного продукту:

а) створення композиції з віджетів, де віджет – це запчастина з яких складається інтерфейс користувача. Віджет необхідно створити таким чином, щоб перевикористання було можливим, тобто зробити публічним, винести в окремий клас та наслідувати від `StatelessWidget` або `StatefulWidget` [28].

б) для управління станом (state managment) можна використати `BLoC` або `Provider`. Обидва підхода запропоновані Google та показали високу ефективність при розробці застосунків. Передбачається, що в майбутньому застосунку буде використано `Provider` як більш зручний у використанні [29].

в) доступ до теми застосунку (тематичні кольори, шрифти, розміри, стилі кнопок тощо), локалізації, інформації про пристрій на якому виконується програма тощо здійснюється за допомогою контексту (`BuildContext`) [30].

г) для впровадження ін'єкції залежності (`Dependency Injection`) використовуються `GetIt`. Це простий локатор послуг для проєктів Dart і Flutter з деякими додатковими перевагами, натхненними `Splat` [31]. Його можна використовувати замість `InheritedWidget` або `Provider` для доступу до об'єктів. Також популярністю користується бібліотека `Injectable`, яка є зручним генератором коду для `GetIt` [32].

д) як було зазначено у другому розділі, об'єкти передачі даних (DTO) необхідні для взаємодії застосунка з сервером, тому їх наявність у проєкті дуже важлива. Flutter надає інструменти для перетворення json-об'єктів у DTO та навпаки.

Кожен Flutter проєкт має чотири головні папки: `android`, `ios`, `lib` та `test` (див. рис. 4.4). У папках `android` та `ios` знаходиться нативний код та, при необхідності, туди можна додати функціональність, яка залежить від нативних

платформ. У папки test знаходяться модульні та інтеграційні тести. Головною директорією є папка lib, яка містить код проєкту. Також слід звернути увагу на файл проєкту pubspec.yaml який відповідальний за версію проєкту, його назву, бібліотеки та файли які використовуються у проєкті.

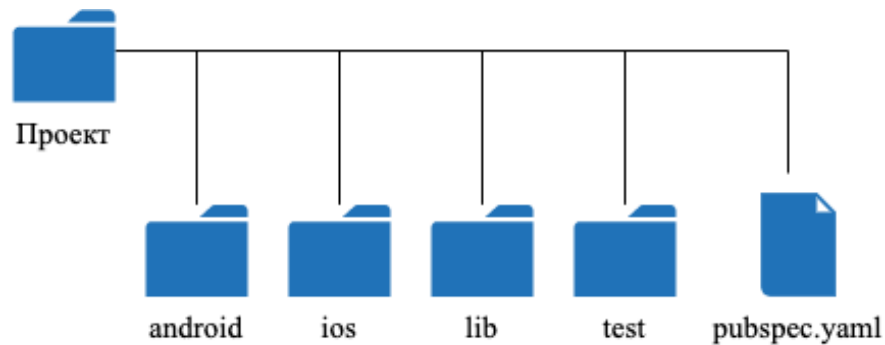


Рисунок 4.4 – Структура папок Flutter проєкту

Як мову програмування для реалізації даного застосунку буде використано Dart, так як це основна та єдина мова програмування фреймворку Flutter. Dart, як і Flutter, було розроблено та підтримується Google. Забігаючи наперед, довгий час, Dart був не затребуваний, але з появою Flutter, мова почала нове життя.

Dart – об’єктно-орієнтована мова. У своєму розвитку вона зазнала впливу більшості ранніх мов, тому її синтаксис схожий на синтаксис інших С-подібних мов таких як Java, JavaScript та C#. Dart швидко розвивається і у березні 2021 року, оновила синтаксис мови, запровадивши нульову безпеку (null safety). На прикладі мов Kotlin та Swift, які давно використовують нульову безпеку, такий підхід позбавив розробників необхідності відловлювати помилки, які спричинені об’єктами зі значенням null [33].

У поточному підрозділі було перелічено основні моменти, які слід враховувати при розробці запланованого програмного продукту та які можуть полегшити процес впровадження архітектурних патернів у проєкт, що розробляється на програмному каркасі Flutter.

4.3 Вибір середовища розробки

Зараз є безліч інструментів які роблять процес розробки швидким та зручним, однак так як програмний каркас було обрано Flutter, необхідно обрати саме тій засіб, який має інтеграцію з цим фреймворком.

Найбільш популярні це Android Studio та VS Code. Однак Android Studio можна класифікувати як інструмент у категорії “Інтегроване середовище розробки”, а VS Code як “Текстовий редактор”.

Далі представлено таблиці 4.2 та 4.3, на яких показано порівняльну характеристику середовищ розробки.

Таблиця 4.2 – Порівняльна характеристика середовищ розробки

Критерій	Android Studio	IntelliJ Idea	VSCode
Тип	Інтегроване середовище розробки	Інтегроване середовище розробки	Текстовий редактор
Дата випуску	грудень 2014	січень 2001	квітень 2015
Вбудований емулятор	є	немає	немає
Попереднє налаштування	Мінімальна, в установник включені необхідні пакети для розробки мобільних застосунків на Android	Необхідно завантажити і підключити необхідні плагіни і бібліотеки	Необхідно завантажити і підключити необхідні плагіни і бібліотеки
Швидкодія (суб’єктивна оцінка за 10-ти бальною шкалою)	5	6	9
Ліцензія	Apache License 2.0	Community: Apache License 2.0 Ultimate: Trialware	MIT License та Freeware

IntelliJ Idea у цієї таблиці не випадково, можна сказати, що це батько AndroidStudio: основний функціонал студія запозичила у ідеї, однак, з часом AndroidStudio обрала свій шлях розвитку та стала незалежною від IntelliJ Idea.

Щодо популярності середовищ розробки, то Asana, Microsoft та Intuit – одні з популярних компаній, які використовують код VS, тоді як Android Studio використовуються Google, Lyft та 9GAG. Код Visual Studio має більш широке схвалення, згадується в 1134 стеках компанії та 2381 стеках розробників; порівняно з Android Studio, який занесений у 928 стек компаній та 692 стеки розробників. Це пов'язано з тим що Android Studio використовується тільки для мобільної розробки, у той час коли VSCode більш універсальний та використовується майже для всіх типів розробки та підтримує безліч мов програмування [34].

Таблиця 4.3 – Порівняльна характеристика мінімальних системних вимог середовищ розробки

Вимоги	Android Studio	IntelliJ Idea	VSCode
Доступне місце на жорсткому диску	2 Гб (0.5 Гб для IDE + 1.5 Гб для Android SDK та емулятора)	2.5 Гб	200 Мб
Оперативна пам'ять	4 Гб + 4 Гб для стабільної роботи емулятору	2 Гб	1 Гб
Розширення екрану	1280 x 800	1024 x 768	800 x 600
Платформи	X Yosemite, 64-бітна версія Windows 10, Linux	X Yosemite, Windows 10, Linux	X Yosemite, Windows 10, Linux

Підводячи підсумок та основууючись на даних двох останніх таблиць, AndroidStudio хоч й потребує більше ресурсів ніж інші, але вона має всі необхідні та зручні інструменти для мобільної розробки. Також AndroidStudio

підтримується Google та рекомендується як основний інструмент для розробки на Flutter.

4.5 Налаштування серверного постачальника послуг

Так як у якості постачальника послуг було обрано Firebase, Cloud Firestore буде використано як БД для програмного продукту. Для зручної авторизації буде додано послугу від Firebase – Firebase Auth, яка надає безліч способів входу у систему, у тому числі за допомогою звичайного логіну-пароллю або через Google акаунт. Для спрощеної авторизації пропонується використовувати Google акаунт, тому що такий підхід буде надійним, безпечним та зручним. На рисунку 4.5 видно, що спосіб авторизації для входу у систему буде здійснюватися тільки за допомогою Google. Для реалізації авторизації знадобиться дві бібліотеки `firebase_auth` та `google_sign_in`.

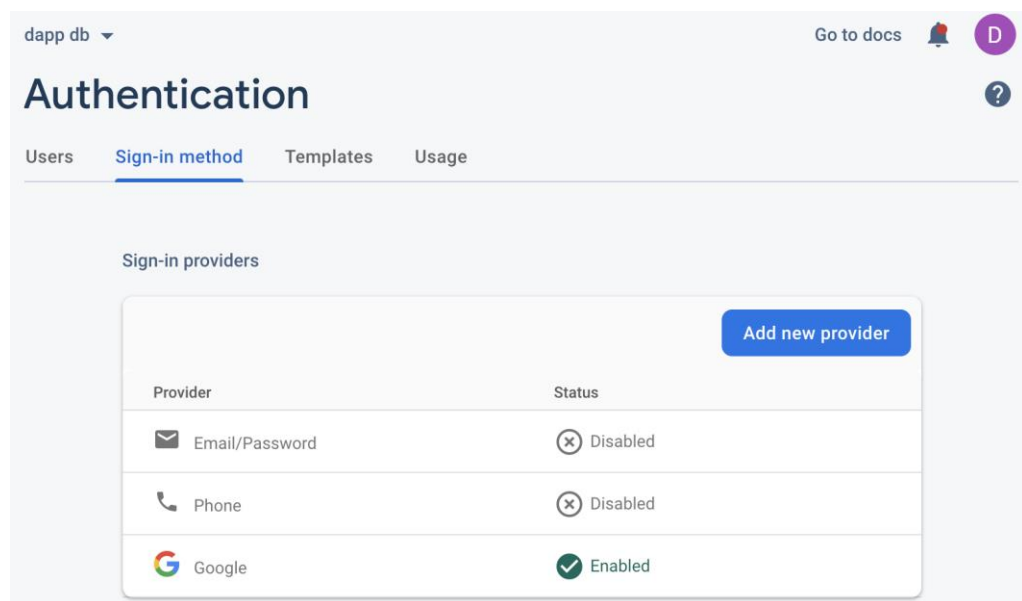


Рисунок 4.5 – Вибір провайдеру для авторизації

Слід відзначити що, для впровадження авторизації за допомогою Google необхідно додати SHA-1 та SHA-256 у налаштуваннях до проекту як показано на рисунку 4.6. однак це необхідно зробити тільки для Android

проєкту. Для цього треба ввести наступну команду у терміналі: `keytool -list -v -alias androiddebugkey -keystore ~/.android/debug.keystore`. Потім утиліта `keytool` запропонує ввести пароль для сховища ключів [35]. Пароль за замовчуванням для сховища ключів налагодження – `android`. Отримані SHA слід додати у Firebase, як це показано на рисунку 4.6.

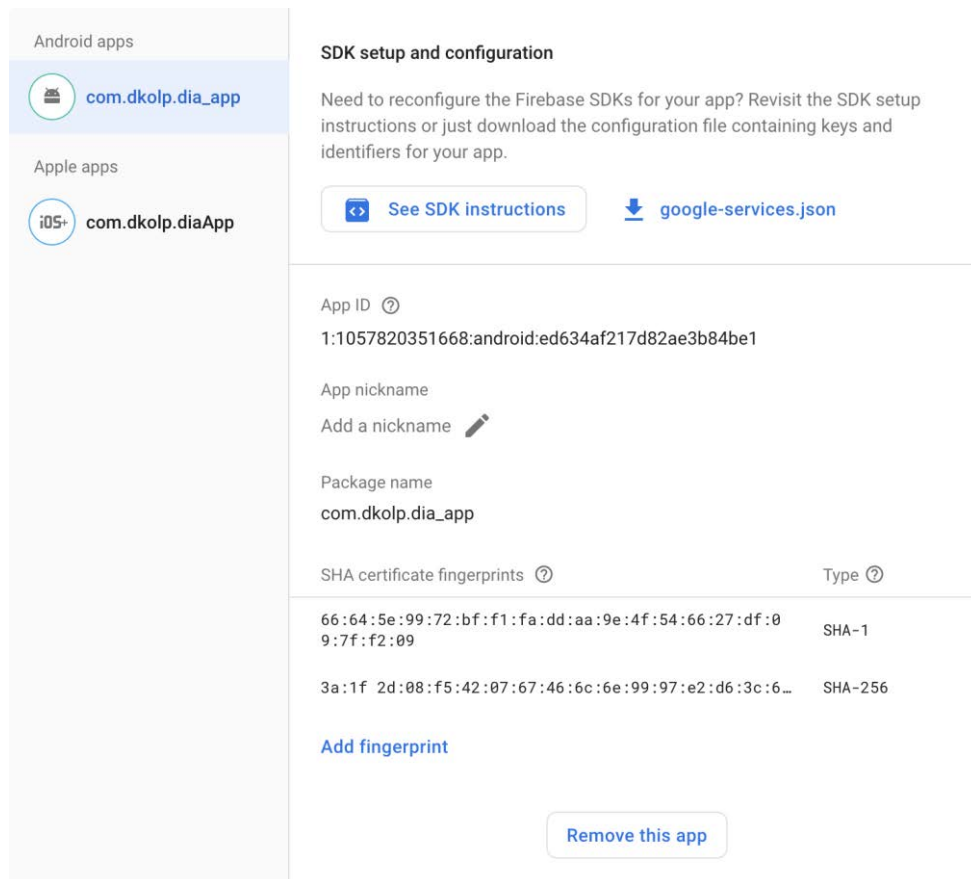


Рисунок 4.6 – Налаштування для Android у Firebase

Головна частина та мета розробки проєкту – база даних. При налаштуванні важливо обрати місцезнаходження БД як показано на рисунку 4.7. Розташування вказується при створенні БД та не може бути змінено у майбутньому. Також вибір місця впливає на швидкість роботи БД: чим ближче до клієнтів, тим краще.

Слід звернути увагу що структура БД створюється клієнтом: саме від нього залежить як вона виглядає. Створену структуру БД можна побачити на рисунку 4.8. Тут згенеровано дві таблиці: лікарі та пацієнти. У Cloud Firebase

таблиці уявляють собою дерево, яке складається з колекцій та документів. У такої структури доступ до даних здійснюється дуже просто та не треба багато замислюватися над отриманням або оновленням даних. Тим не менш, використання NoSQL, має свої недоліки. У випадку Firebase, це неможливість створення складних запитів з необмеженою кількістю умовностей. Є шляхи для обходу цього обмеження, однак це завдає багато незручностей.

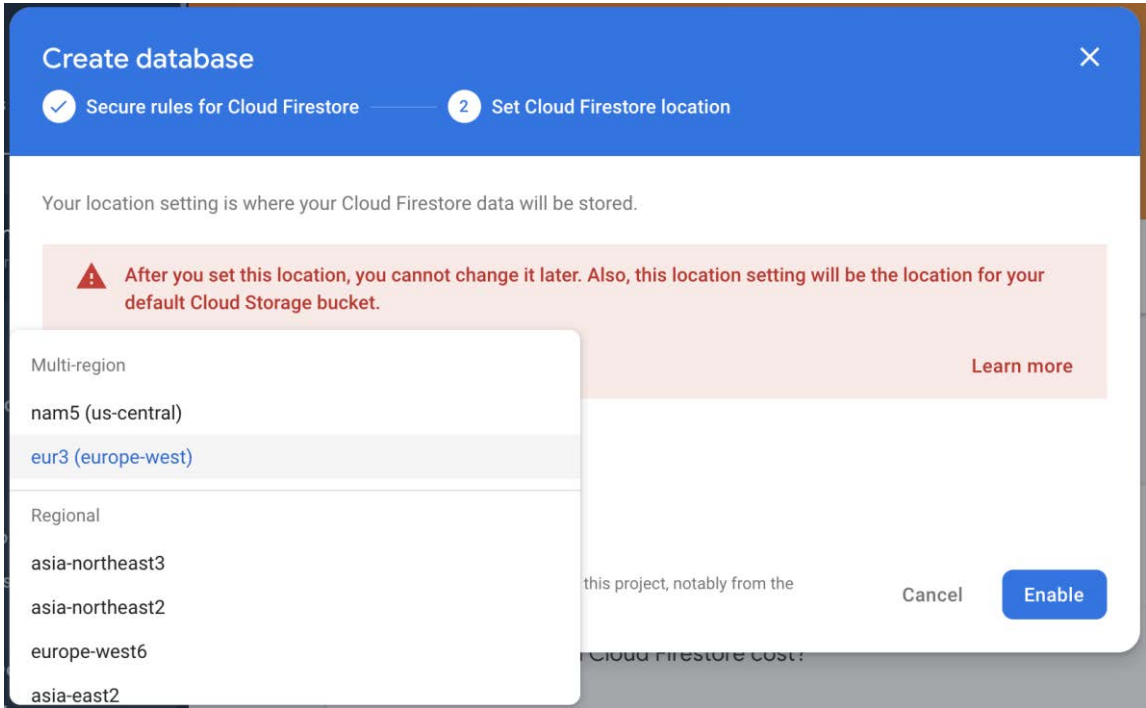


Рисунок 4.7 – Налаштування місцезнаходження БД

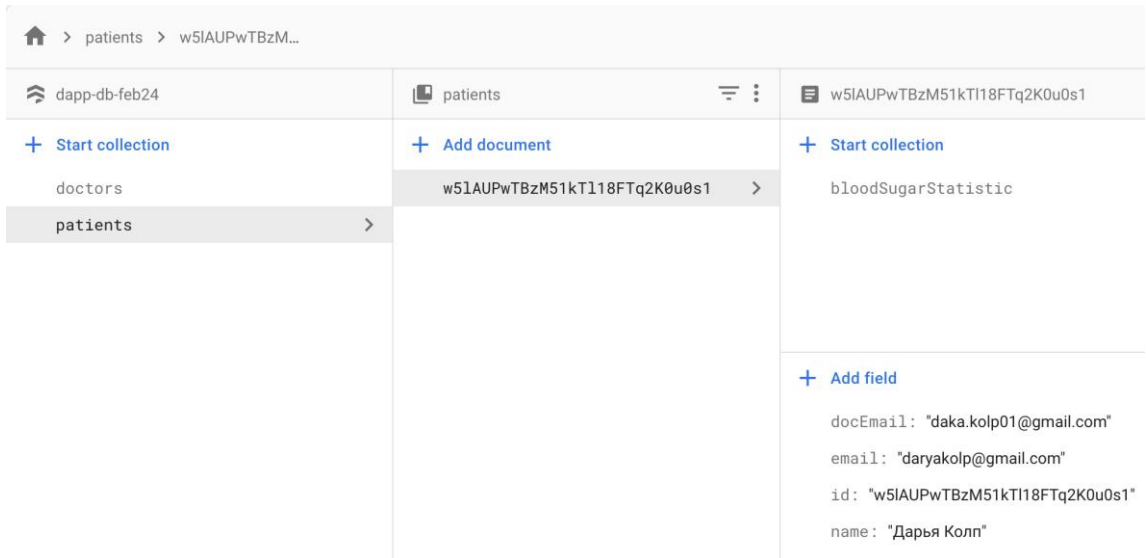


Рисунок 4.8 – Таблиця пацієнтів, яка представлена у базі даних Firestore

У першій зберігаються необхідні дані лікарів. Кожен з лікарів має список ідентифікаторів користувачів, які погодились на перегляд даних лікарем про стан їх здоров'я. Це надає змогу лікарю, стежити за станом здоров'я пацієнтів дистанційно у будь-який час, без турбування хворого на діабет. У таблиці з пацієнтами зберігається необхідна особиста інформація про пацієнта, та статистичні дані про стан цукру у його крові за весь час користування застосунком. На рисунку 4.9 відображено приклад такої статистики.

w5IAUPwTBzM51kTI18FTq2K0u0s1	bloodSugarStatistic	1619426705133
+ Start collection	+ Add document	+ Start collection
bloodSugarStatistic >	1619426705133 >	+ Add field
+ Add field	1619426826916	bloodSugar: 2.6
docEmail: "daka.kolp01@gmail.com"	1619458811513	timeOfMeasure: April 26, 2021 at 11:45:05 AM UTC+3
email: "daryakolp@gmail.com"	1619458815619	
id: "w5IAUPwTBzM51kTI18FTq2K0u0s1"	1619458819708	
name: "Дарья Колп"	1619543195892	
	1619543203265	
	1619543225693	

Рисунок 4.9 – Таблиця зі статистичними даними про стан цукру у крові у пацієнта

Слід зауважити, що база даних Firebase часто використовується для стартапів та має залежність від платформи, тому необхідно втручатися у нативний код та робити налаштування для кожної платформи згідно з документацією Firebase. Наприклад, для коректної роботи застосунку у модуль Android потрібно додати google-services.json, а у модуль iOS – GoogleService-Info.plist. На щастя, архітектура проєкту побудована таким чином, що надає змогу швидко змінювати логіку роботи з базою даних без втручання в код, який відповідальний за інтерфейс користувача.

4.5 Розробка програмного продукту

Особливу увагу слід приділити структурі папок проєкту, адже грамотне розташування папок допомагає простіше орієнтуватися в проєкті та наочно побачити архітектуру проєкту (див. рис. 4.10).

Проєкт складається з основного каталогу вихідних файлів та головного файлу `main.dart`. Каталог вихідних файлів поділено на чотири головні папки, кожна з яких відповідає за окремий шар програми.

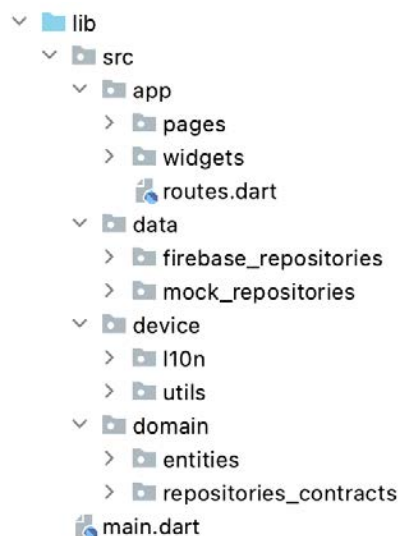


Рисунок 4.10 – Структура каталогів проєкту

а) `domain`. Є серцем проєкту, та визначає сутності та бізнес-логіку застосунку. Класи у цьому каталозі самостійні та не мають жодної залежності від коду, який знаходиться поза його межами.

б) `data`. Тут знаходяться реалізації репозиторіїв, які було визначено у каталозі `domain`. На рисунку 4.10 видно, що тут розташовані, як реальна реалізація методів API постачальника серверних послуг Firebase, так і фіктивна – `Mock`. Друга може знадобитися для модульного тестування та для розробки, щоб не витрачати ресурси платформи у процесі створення інтерфейсу користувача.

в) `app`. Зберігає у собі класи, які відповідають за створення інтерфейсу користувача та за навігацію між сторінками. Вони можуть мати інформацію

про класи з каталогу domain, а саме про бізнес-логіку застосунку, так як це не заперечує принципу чистої архітектури.

г) device. Містить у собі допоміжні класи, наприклад такі як локалізація застосунку або утиліта для доступу внутрішнього сховища.

Як можна побачити, структура доволі проста та опосередковано визначає архітектурне рішення, яке було прийнято для розробки застосунку. Далі описано повноцінне створення застосунку.

Питання, з чого треба почати розробку програмного забезпечення, з інтерфейсу користувача або логіки, завжди постає на початку розробки. Однак, очевидно, що реалізація бізнес-логіки є першочерговим завданням. Тому слід почати з реалізації сутностей проєкту. Важливо, щоб сутності визначали застосунок, а не дані які зберігаються на сервері. Не здатність побачити різницю між ними – це основне питання та проблема, яку ігнорують розробники програмного забезпечення. В поточному проєкті основними сутностями є користувач та статистика (див. рис 4.11).

```
abstract class User {
    final String id;
    final String email;
    final String fullName;
    User({
        @required this.id,
        @required this.email,
        this.fullName,
    }) : assert(id != null), assert(email != null);
}

class Patient extends User {
    final String docEmail;
    Patient({String id, String email, String fullName, this.docEmail})
        : super (id: id, email: email, fullName: fullName);
}

class Doctor extends User {
    final List<String> patientIds;
    Doctor({String id, String email, String fullName, this.patientIds})
        : super (id: id, email: email, fullName: fullName);
}
```

Рисунок 4.11 – Сутність User та її дочірні класи

Не має значення, що повертає сервер, це може бути купа марних даних, важливо лише те, що потрібно застосунку для реалізації вимог до проєкту. Як можна побачити на рисунку 4.11, клас User має лише три властивості, а класи

що успадковуються від `User`, мають ще по одній. Це обумовлено тим, що на поточному етапі розробки, цих даних достатньо для виконання вимог проєкту. Додавання властивостей з гіпотетичною можливістю використання – це марнування часу, тому що велика ймовірність того, що цей код залишиться мертвим. Тому краще, додавати нові поля у клас у разі необхідності.

Так як планується, що застосунок буде працювати з двома видами користувача, для зручності було створено перелічуваний тип даних: `UserType`. На даний момент надає два типи користувачів: пацієнт та лікар.

Також створено абстрактний клас `Statistic` та `BloodSugarStatistic`, який успадковується від `Statistic`. Це здійснено з тією метою, що у майбутньому застосунок може обробляти не лише дані про цукор у крові, а й інші показники. Тому `Statistic` має єдину властивість, яка буде справедливою для всіх типів статистичних даних, а саме дані про дату та час коли було знято показники. Також додано метод `compareTo`, який зручний для сортування за часом.

У попередньому підрозділі було зазначено, що застосунок повинен надавати змогу швидко змінювати логіку роботи з сервером, а також бути модульним для тестування. Саме тому частина, яка відповідає за роботу з сервісами платформи, повинна залежати від абстрактного класу або інтерфейсу.

Основними та головними класами логіки застосунку є `UserRepositoryFactory`, `UserRepository` та похідні від нього класи: `PatientRepository` та `DoctorRepository`. Весь проєкт має залежність від інтерфейсів `PatientRepository`, `DoctorRepository` та `UserRepository`, де останній створено з метою, щоб не дублювати методи, які необхідно реалізувати у `PatientRepository` та `DoctorRepository`. Ці класи є абстрактними, а їх імплементація залежить від того яку платформу сервісних послуг обрано, тому похідні від них класи можна замінити на будь-яку реалізацію їх інтерфейсів.

Слід зауважити, що доступ до класів-репозиторіїв здійснюється за принципом ін'єкції залежності (`Dependency Injection`). На рисунку 4.12

зображено головний метод проєкту – `main.dart`, з якого починається робота будь-якої програми на мові Dart. Тут можна побачити єдину згадку про реалізовані від інтерфейсів класи, які було описано раніше. Напрямую, в програмному коді, вони не використовуються.

```
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  GetIt.I
    ..registerSingleton<GoogleService>(GoogleService())
    ..registerSingleton<StoreInteractor>(StoreInteractor())
    ..registerSingleton<AuthRepository>(FirebaseAuthRepository())
    ..registerSingleton<UserRepositoryFactory>(FirebaseUserRepositoryFactory());
  final userType = await GetIt.I<AuthRepository>().init();
  runApp(MyApp(userType: userType));
}
```

Рисунок 4.12 – Головний метод запуску програми

Очевидно, що у методі `main()` не вистачає реєстрації `PatientRepository` та `DoctorRepository`. За це відповідає `UserRepositoryFactory`. Від реалізує патерн проєктування “Фабрика”, що реєструє відповідний репозиторій в залежності від того який тип користувача авторизовано у системі. У свою чергу `AuthRepository`, використовує `UserRepositoryFactory` під час авторизації та під час виходу з системи. В методах абстрактного класу `AuthRepository`, реалізовано реєстрацію та виділення репозиторіїв з локатору послуг `GetIt`. Приклад реєстрації відповідного репозиторію користувача в `AuthRepository` наведено на рисунку 4.13, де особливу увагу привертають два чинника.

По перше, можна переконатися, що в `AuthRepository` немає жодної згадки про інтерфейси, а тим більш про реалізовані класи від `UserRepository`, `PatientRepository` та `DoctorRepository`, цю працю виконує `UserRepositoryFactory`.

```

@mustCallSuper
Future<void> loginWithGoogleAsPatient() async {
  if (await isLoggedIn) {
    GetIt.I.registerSingleton<UserRepository>(
      GetIt.I<UserRepositoryFactory>().createUserRepository(
        UserType.Patient,
      ),
    );
  }
}

```

Рисунок 4.13 – Метод в абстрактному класі AuthRepository

Для того щоб створити відповідні класи, фабрики потрібно знати лише тип користувача, `UserType`, про який згадувалося раніше. У випадку якщо тип користувача не відомий, метод `createUserRepository` поверне помилку про не існуючий тип користувача.

По друге, методи, які працюють разом з `GetIt` та `UserRepositoryFactory`, мають анотацію `@mustCallSuper`, яка змушує дочірні класи викликати методи батьківського класу. Таким чином, було створено безпечне розділення системи на застосунок для пацієнта та застосунок для лікаря: `PatientRepository` та `DoctorRepository` не можуть існувати одночасно, що виключає виклик методів, які не призначені для використання конкретним типом користувача.

Що стосується `AuthRepository`, то він має лише 2 геттера, які надають статус користувача, що авторизувався та 4 методи, які відповідають за авторизацію за допомогою Google та виходу з системи. `FirebaseAuthRepository` успадковується від `AuthRepository` та реалізує його методи. Весь проєкт має залежність тільки від `AuthRepository`, тому `FirebaseAuthRepository` можна замінити на будь яку іншу реалізацію `AuthRepository`, як і планувалося.

Авторизацію через Google імплементовано у реалізації абстрактного класу `AuthRepository` – `FirebaseAuthRepository`. Для реалізації було використано `FirebaseAuth` та `GoogleSignIn`, які згадувалося у попередньому підрозділі.

Для зручності, функціональність роботи з авторизацією через Google, було винесено у окремий клас `GoogleService`, як показано на рисунку 4.14. Ця логіка самостійна, та може бути використана для реалізації входу до

особистого кабінету за допомогою Google для будь-якого постачальника сервісних послуг. Однак не слід забувати про налаштування, про які згадувалося у попередньому підрозділі, а саме необхідно додати SHA-1 та SHA-256 у налаштуваннях до проєкту для Android, тому що це справедливо для будь-яких бекенд-сервісів які підтримують авторизацію за допомогою Google.

```
class GoogleService {
    final _googleSignIn = GoogleSignIn(scopes: <String>['email']);
    Future<GoogleSignInAuthentication> getGoogleAuthData() async {
        final account = await _googleSignIn.signIn();
        final isSignedIn = await _googleSignIn.isSignedIn();
        if (isSignedIn) {
            try {
                return await account.authentication;
            } finally {
                await _googleSignIn.disconnect();
            }
        }
        throw GoogleLoginException();
    }
    Future<void> signOut() async {
        await _googleSignIn.signOut();
    }
}
```

Рисунок 4.14 – Реалізація допоміжного класу для авторизації через Google

На основі спрощеної діаграми класів, яку було запропоновано у третьому розділі, реалізовано класи, які надають функціональність сценарію використання (Use Case), назву яких було запроваджено як Presenter. Для зручності він успадковується від ChangeNotifier-y. ChangeNotifier це простий клас, що входить до програмного каркасу Flutter, який забезпечує сповіщення про зміни слухачам. Іншими словами, якщо щось є ChangeNotifier, то можна підписатися на його зміни. Таким чином Presenter не знає про те, який клас, що відповідає за представлення, його використовує, але знає про інтерфейси репозиторіїв. Він дозволяє представленню відстежувати, коли слід оновлюватися за допомогою методу notifyListeners() та використовує методи репозиторію. Приклад коду на рисунку 4.15, показує як у представленні ChangeNotifierProvider вказує який саме Presenter воно використовує. Класи

Presenter-у знаходяться у тому ж каталозі, що й класи інтерфейсу користувача.

```
static Widget _builder(BuildContext context) {
  return ChangeNotifierProvider(
    create: (context) => MainPagePresenter(context),
    child: MainPage(),
  );
}
```

Рисунок 4.15 – Приклад ініціалізації Presenter-у

В цілому, проєкт містить 5 класів представлень, два з котрих реалізовано для різних типів користувачів, та стільки ж класів сценарію використання, які діють за однаковим механізмом роботи, який було описано раніше (див. рис. 4.16).

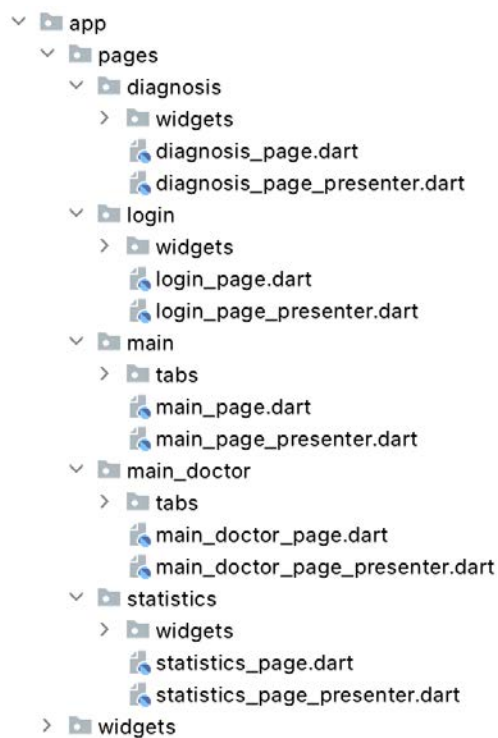


Рисунок 4.16 – Структура каталогів та класів, які відносяться до пласту представлення застосунку

Для впевненості у коректній роботі застосунку та його модулів написано UNIT-тести. За допомогою створених фіктивних репозиторіїв

тестується робота методів. Тестування класів-репозиторіїв є можливим завдяки абстрагуванню від реалізації, про що було згадано раніше. Результати модульного тестування фабричного класу `UserRepositoryFactory` та репозиторіїв `UserRepository` і `AuthRepository` описано у п'ятому розділі, для цього було створено реалізації з префіксом `Mock` для імітації роботи сервером.

Крім того, реалізовано тести для сутностей проєкту: `User` та `Statistic`. Так як саме вони визначають бізнес-логіку застосунку, важливо забезпечити їм покриття тестами. Це обумовлено тим, що скоріше за все, проєкт буде розширюватися та буде реалізовувати нові сутності, репозиторії та функціональність в цілому. У процесі розробки часто можна щось не помітити, видалити або змінити, результатом чого можуть бути неочевидні помилки у проєкті. Їх можна не помітити під час ручного тестування, але вони можуть проявити себе на етапі експлуатації. Тому таке тестування необхідне, бо може своєчасно виявити помилки на етапі розробки, що надає можливість їх швидко виправити. Так як у проєкті є два типи користувача, важливо було реалізувати поліморфну структуру, при якому лікар та пацієнт відносяться до одного виду, а саме користувача. Код для тестування `Statistic` складніший ніж для сутності користувача, та, серед іншого, перевіряє чи правильно визначає діагноз сутність `BloodSugarStatistic`, основуючись на показнику кількості цукру в крові.

Повний текст програмної реалізації та модульних тестів знаходиться у додатку А.

4.6 Висновок до четвертого розділу

У четвертому розділі було обґрунтовано вибір програмного каркасу `Flutter` та середовища розробки `AndroidStudio`, як основного інструменту для розробки на `Flutter`. Розглянуто структуру бази даних проєкту створеної у `Cloud Firestore`, а також `GoogleAuth`, як спосіб авторизації користувачів у системі `Firebase`. Докладно описано розробку застосунку та інтеграцію

Firebase сервісів у нього. Звернено увагу на важливі аспекти реалізації кросплатформного мобільного застосунку. Реалізовано проєкт з використання адаптивного архітектурного шаблону який було докладно описано у третьому розділі. Також було написано UNIT-тести для репозиторіїв та сутностей програми.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

У поточному розділі слід протестувати систему, яку було розроблено у четвертому розділі. Тестування повинно поєднувати у собі як тестування використаних сервісів платформи, так і перевірку працездатності розробленого застосунку.

5.1 Тестування Cloud Firestore

Cloud Firestore має правила безпеки, які контролюють доступ до документів і колекцій у БД. Гнучкий синтаксис правил дозволяє створювати правила, які відповідають будь-якому, від усіх записів до всієї бази даних до операцій над певним документом. За замовченням, в залежності від того яким чином реєструється БД, код сценарію може відрізнятися. Наприклад, якщо БД створюється у виробничому режимі, то сценарій робить всі дані у БД приватними, як показано на рисунку 5.1. У випадку якщо БД створюється у тестовому режимі, сценарії надає відкритий доступ до даних на 30 днів [36].

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if false;
    }
  }
}
```

Рисунок 5.1 – Приклад сценарію правил використання Cloud Firestore

Для впевненості, що база даних що Cloud Firestore, працює правильно з дотриманням вимог до безпеки даних, слід змінити сценарії правил роботи БД та протестувати їх роботу.

Для початку, щоб продемонструвати як працює сценарій, його код можна модифікувати таким чином, щоб дозволяти зареєстрованим користувачам мати доступ до БД та заборонити незареєстрованим читати або записувати дані (див. рис. 5.2).

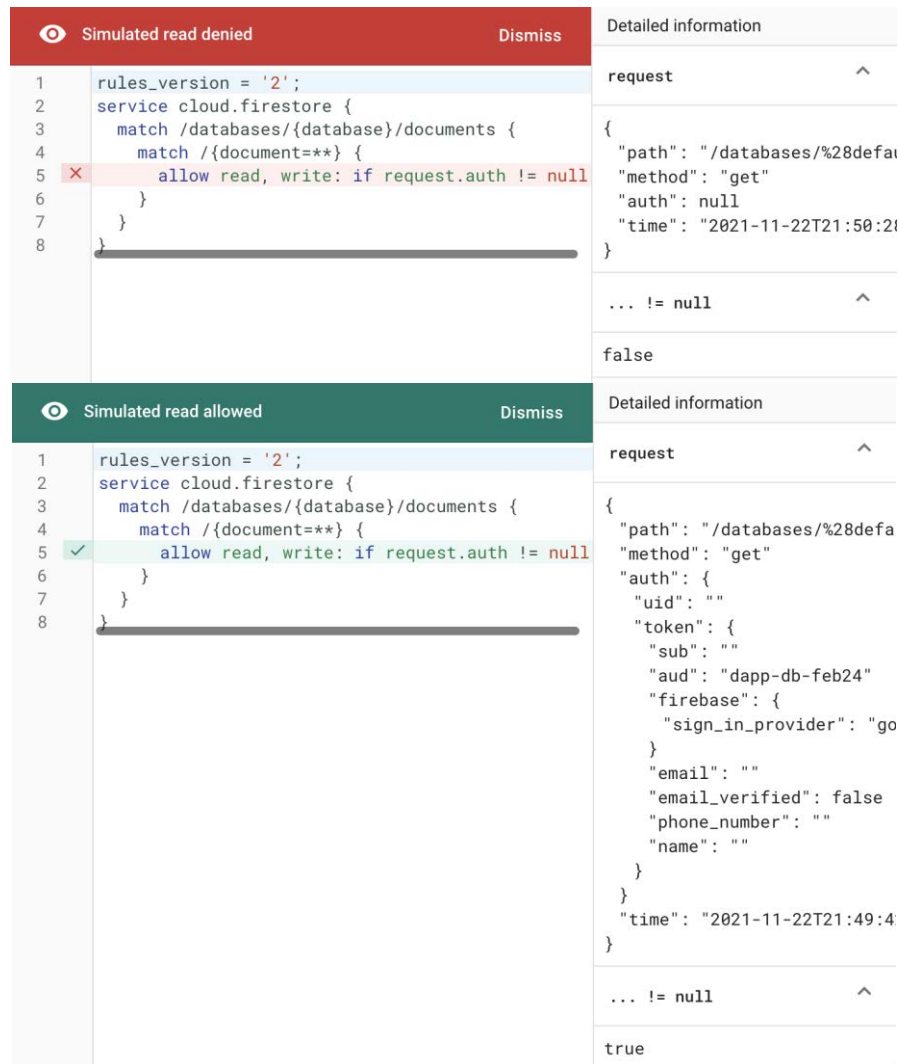


Рисунок 5.2 – Результат тестування сценарію, який дозволяє лише зареєстрованим користувачам мати доступ до БД Cloud Firestore

Даний сценарій перевіряє чи має доступ до бази даних неавторизований користувач. Цей код можна вдосконалити, таким чином, щоб користувач з відповідним ідентифікатором мав доступ у Cloud Firestore до документів, які відносяться до цього ідентифікатора, та не мав доступу до інших. Такий код буде мати вид як на рисунку 5.3.

```

service cloud.firestore {
  match /databases/{database}/documents {
    match /patients/{userId}/{document=**} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
  }
}

```

Рисунок 5.3 – Сценарій, при якому зареєстрований пацієнт має доступ тільки до особистих даних у БД Cloud Firestore

Використання сценаріїв дозволяє забезпечити безпеку даних, які зберігаються у БД, тому їх тестування є важливим під час процесу впровадження Cloud Firestore у проєкт.

Далі на рисунку 5.4. зображено графік, який демонструє роботу створених правил. Під час тестування протягом тижня, не було жодних помилок або заперечень, що свідчить про те що правила працюють, а застосунок спроектовано таким чином, що відповідає цим правилам.

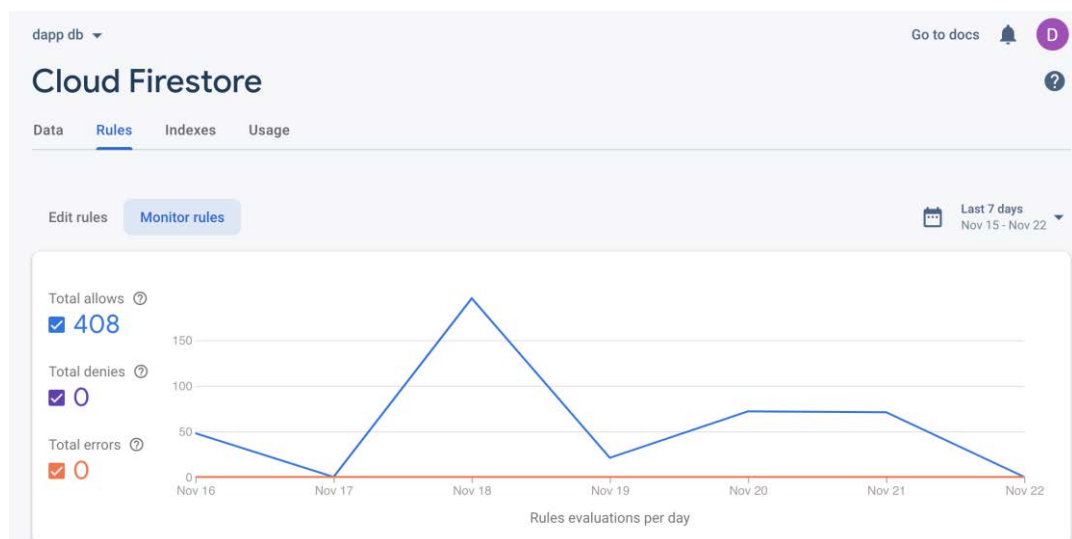


Рисунок 5.4 – Графік, який демонструє роботу створених сценаріїв

Також Firebase має інструменти для спостереження за кількістю зчитувань, записів та видалень з БД (див. рис. 5.5), та для моніторингу кількості зареєстрованих та активних користувачів. Це важливо, враховуючи

тій факт, що безкоштовний тариф Firebase лімітований та має свої квоти. Перелік використаних квот на поточному проєкті, наведено на рисунку 5.6.

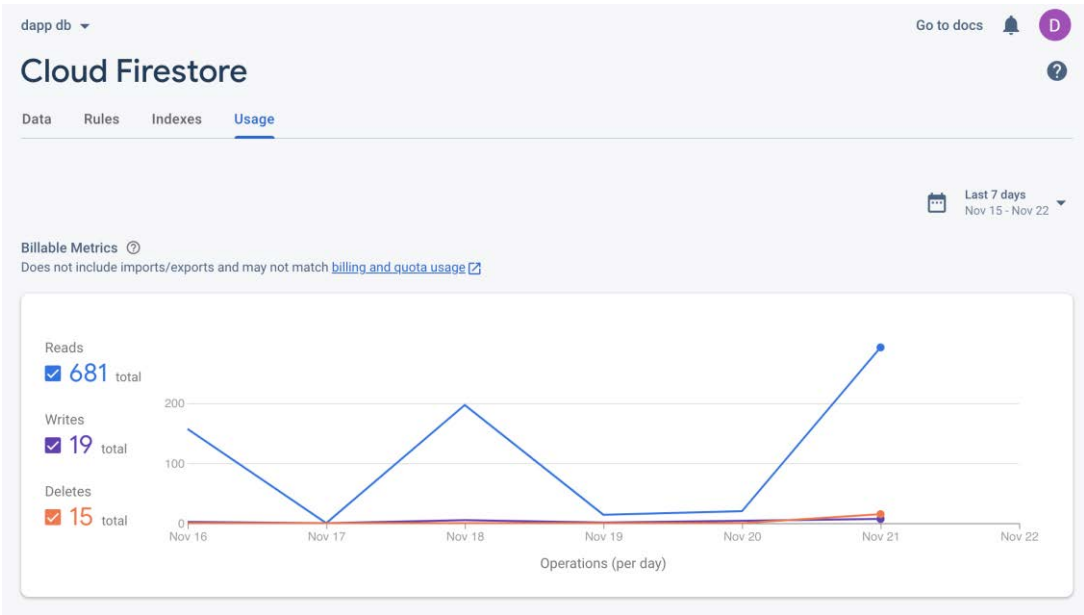


Рисунок 5.5 – Графік кількості зчитувань, записів та видалень БД

Storage	
Resource	Usage today
Cloud Firestore Read Operations	0.000012 of 0.05 Million Ops
Cloud Firestore Small Operations	0.000002 of 0.05 Million Ops
Cloud Firestore API Calls	26
Cloud Firestore Stored Data	0.000026 of 1 GB
Cloud Firestore Entity Fetch Ops	9
Cloud Firestore Entity Writes	0.000001 of 0.02 Million Ops
Cloud Firestore Index Write Ops	7
Cloud Firestore Query Ops	3
Cloud Firestore Key Fetch Ops	2
Cloud Firestore Network Egress	0.000003 GB

Рисунок 5.6 – Перелік використаних квот під час тестування

Очевидно, що під час тестування не було використано 1% від наданих Firebase квот. Тобто для запуску нового програмного продукту дана платформа вигідна та, при невеликій кількості користувачів, є рентабельною.

5.2 Тестування взаємодії застосунку і сервісів Firebase

Щоб впевнитися, що додавання даних у застосунку працює та дані зберігаються на сервері, необхідно використати реалізацію репозиторіїв, які мають доступ до бекенд як сервісу, та провести ручне тестування. У попередньому розділі було реалізовано код, який працює з сервісами Firebase, тому далі буде демонструватися робота з ними.

Основна функція програмного продукту це зберігання статистики цукру в крові для пацієнтів та перегляд статистичних даних пацієнтів для лікаря. Далі наведено три сценарії роботи з застосунком та зображення сторінок, які доводять, що програма та сервер взаємодіють правильно. По-перше, слід протестувати функціональність для авторизації.

Крок 1. Натискаємо на кнопку “Зайти за допомогою Google” або на кнопку “Зайти за допомогою Google як лікар”.

Крок 2. У відкритому вікні обрати Google акаунт, за допомогою якого система авторизує або реєструє користувача (див. рис. 5.7).

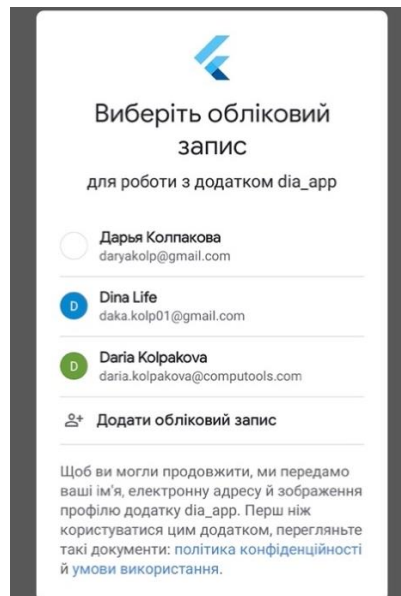
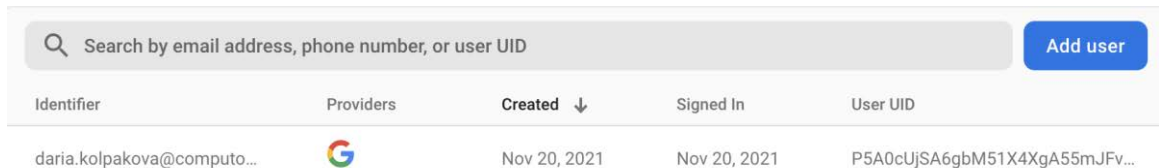


Рисунок 5.7 – Діалогове вікно для авторизації за допомогою облікового запису Google

Для тестування обрано обліковий запис Google з поштою daria.kolpakova@computools.com.

Крок 3. Користувач з поштою daria.kolpakova@computools.com не авторизований, тому у Firebase можна перевірити, чи була автоматична реєстрація успішною. Як видно на рисунку 5.8 обліковий запис створено та зареєстровано 21 листопада 2021 року.



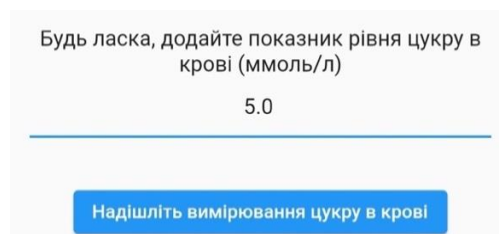
Identifier	Providers	Created ↓	Signed In	User UID
daria.kolpakova@computo...		Nov 20, 2021	Nov 20, 2021	P5A0cUjSA6gbM51X4XgA55mJFv...

Рисунок 5.8– Зареєстрований користувач від 21.11.2021р. у сервісі Auth
Firebase

Таким чином, можна переконатися, що процес авторизації та реєстрації працює коректно, з дотриманням вимог до проекту.

Далі розглянемо другий сценарій, який описує процес додавання даних про рівень глюкози у крові.

Крок 1. На головній сторінці для пацієнта, введемо дані про рівень цукру в крові (див. рис. 5.9) та натискаємо кнопку “Надішліть вимірювання цукру в крові”. Після чого отримуємо повідомлення представленому на рисунку 5.10.



Будь ласка, додайте показник рівня цукру в крові (ммоль/л)

5.0

Надішліть вимірювання цукру в крові

Рисунок 5.9– Приклад введення даних про рівень глюкози



Дані збережено: 20 лист. 2021 17:07, 5.0 ммоль/л,
Норма

Рисунок 5.10– Повідомлення про успішне збереження даних

Крок 2. Переконаємося, що дані, з відповідною інформацією, збережено у Cloud Firestore. Як можна побачити на рисунку 5.11 дані дійсно збережені та ідентичні повідомленню на рисунку 5.10: рівень цукру дорівнює 5 ммоль/л, показник знято 5:07 вечора 21 листопада 2021 року.

```
bloodSugar: 5
timeOfMeasure: November 20, 2021 at 5:07:30 PM UTC+2
```

Рисунок 5.11 – Збережені дані про стан цукру у крові у Cloud Firestore

Крок 3. Перевіримо, чи дані відображаються на сторінці “Статистика” у застосунку. Згідно рисунку 5.12, дані відповідають даним які збережено у Cloud Firestore.

5.0 ммоль/л 20 лист. 2021 17:07	Норма
------------------------------------	-------

Рисунок 5.12 – Дані у застосунку, які було збережено

Проведений експеримент довів, що дані зберігаються у хмарній БД та правильно відображаються у застосунку.

Останній сценарій продемонструє роботу застосунку з типом користувача “Лікар”. Основною вимогою для лікаря це додавання клієнтів до списку пацієнтів та перегляд даних про їх стан.

Крок 1. На головній сторінці для лікаря, для додавання клієнта у список пацієнтів, вводимо ідентифікатор користувача (див. рис. 5.13).

P5A0cUjSA6gbM51X4XgA55mJFvK2

Додати пацієнта

Дарья Колп
daryakolp@gmail.com

Рисунок 5.13 – Додавання пацієнта у список лікаря

Ідентифікатор, який був наданий пацієнтом, має значення P5A0cUjSA6gbM51X4XgA55mJFvK2 та відповідає користувачу, що був зареєстрований при проходженні першого сценарію.

Крок 2. Перевіримо, чи ідентифікатор клієнта збережено у список пацієнтів лікаря у Cloud Firestore. На рисунку 5.14 видно, що до списку додано пацієнта з ідентифікатором P5A0cUjSA6gbM51X4XgA55mJFvK2.

```
email: "daka.kolp01@gmail.com"
id: "gBu910cUGqTAHLB5JOwGTd9OSsJ2"
name: "Dina Life"
▼ patients
  0 "w5IAUPwTBzM51kTI18FTq2K0u0s1"
  1 "P5A0cUjSA6gbM51X4XgA55mJFvK2"
```

Рисунок 5.14 – Дані лікаря зі списком ідентифікаторів пацієнтів у Cloud Firestore

Також у застосунку з’явився відповідний пункт у списку клієнтів, що відповідає доданому пацієнту (див. рис. 5.15).

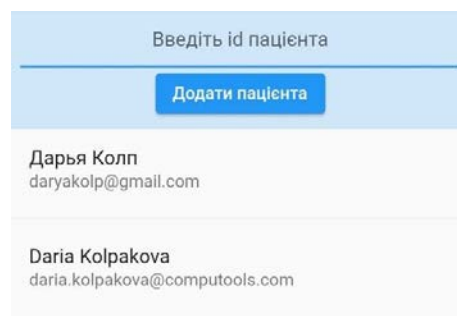


Рисунок 5.15 – Пацієнт, доданий у список лікаря

Крок 3. Перевіримо, чи є релевантними дані про доданого пацієнта. Для цього натиснемо на пункт, який відповідає користувачу з поштою daria.kolpakova@computools.com, після чого буде відкрито сторінку, зображення якої наведено на рисунку 5.16.

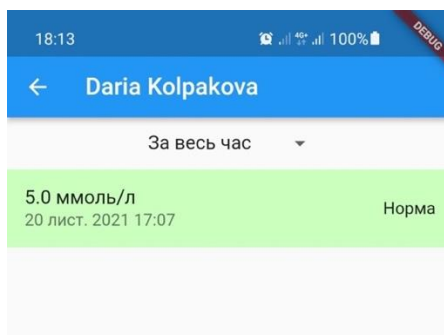


Рисунок 5.16 – Сторінка пацієнта

Відображені дані відповідають даним користувача, котрого було зареєстровано у першому сценарії та дані якого додано у другому.

Підводячи підсумок, провівши три експерименти з реєстрацією користувача, збереженням та отриманням даних з хмарної бази даних, можна стверджувати, що застосунок успішно взаємодіє з бекенд-сервісом, а розроблений програмний продукт відповідає очікуванням.

5.3 Огляд інтерфейсу користувача розробленого застосунку

Встановлення програми на ОС Android здійснюється шляхом запуску .apk файлу або через завантаження через Play Маркет. Для iOS процес встановлення буде складніше тому, що розповсюдження застосунку здійснюється лише через AppStore або TestFlight, причому останній використовується тільки для тестування та потребує встановлення додаткового програмного забезпечення на пристрій. Обидва методи є платними так як розробник повинен бути зареєстрованим на ресурсі developer.apple.com та кожного року сплачувати близько 100\$.

Після встановлення застосунку, при першому запуску програми, користувач побачить сторінку авторизації де він може авторизуватись. Реєстрація не передбачена, тому що для входу у систему використовується Google Sign In. Такий підхід є зручним, так як користувачам не потрібно запам'ятати пароль від облікового запису. Також Google акаунт надає ім'я та

електрону адресу користувача, тому система автоматично зареєструє нового користувача з його особистими даними.

Далі, в залежності від того хто зайшов у систему, лікар або пацієнт, застосунок буде мати вигляд, як показано на рисунку 5.17. Як можна побачити, пацієнту надано форму для вводу даних, сторінку для перегляду статистичних даних про стан цукру в крові, а також сторінку профілю пацієнта. Для лікарів немає ніяких зайвих функцій, лише список користувачів системи, які погодилися на спостереження їх стану, форма для додавання пацієнтів до списку лікаря та особистий кабінет.

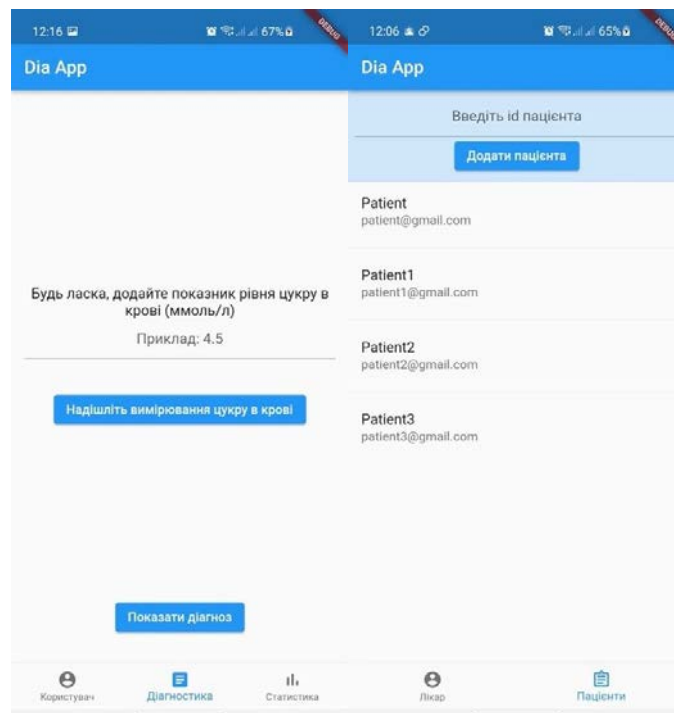


Рисунок 5.17 – Головна сторінка застосунку для пацієнтів (ліворуч) та для лікарів (праворуч)

Використовуючи програмний засіб як пацієнт, головною сторінкою застосунку є “Діагностика”, де користувачу пропонується зберегти дані, за допомогою форми вводу. Валідація не дозволяє завантажити дані до БД, якщо поле пусте або якщо введені значення менше 0 та більше 70, тому що такі показники вважаються безглуздими. Також текстове поле підтримує введення тільки цифр, щоб уникнути очевидних помилок. На рисунку 5.18 представлено

приклад, як текстове поле інформує користувача, про те що дані не відповідають вимогам.

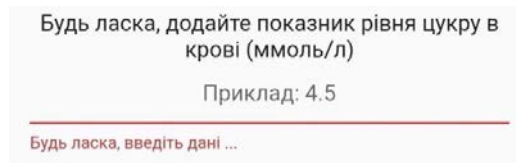


Рисунок 5.18 – Приклад повідомлення текстового поля

Далі, після того, як дані було введено, користувач може отримати одне з двох повідомлень внизу екрану, приклади яких наведено на рисунку 5.19.

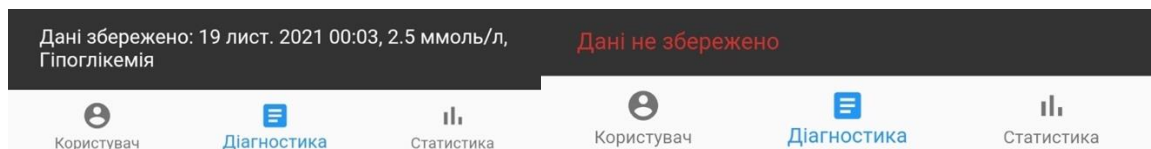


Рисунок 5.19 – Приклад повідомлення після того, як користувач відправив дані до БД

Кнопка “Показати діагноз”, відкриває сторінку діагностики (див. рис. 5.20), де користувач бачить графік статистики показників цукру у крові за обраний період: “За сьогодні”, “За останні 7 днів”, “За останні 30 днів”, “За рік” та “За весь час”.

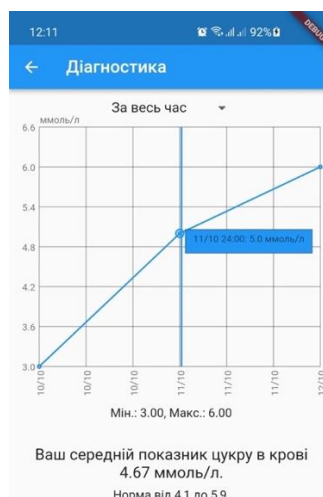


Рисунок 5.20 – Сторінка діагностики

Зручність графіку полягає у тому, що він демонструє загальне уявлення стану здоров'я користувача. Також він надає можливість отримати точну інформацію про вимірювання лише натиснувши на точку на графіку, яка відповідає за показник цукру в крові, який було знято у конкретний час. Тут також надано інформацію про середній показник цукру у крові за обраний період та пам'ятка про значення яке є нормою.

Інтерфейс програми зручний та інтуїтивно зрозумілий. В інтерфейсі використовуються звичайні елементи керування. На головному екрані, перехід між сторінок здійснюється за допомогою кнопок на нижній панелі навігації.

Після натискання кнопки на панелі навігації “Користувач”, користувач бачить екран з особистими даними, які може змінити за необхідністю (див. рис. 5.21). Якщо при редагуванні дані не відповідають формату, користувач отримує відповідне повідомлення. Користувач має можливість відправити особистий ідентифікатор лікарю, щоб той додав пацієнта до свого списку осіб, стан здоров'я яких від відстежує. Також на екрані є кнопка виходу з акаунту, яка розташована у правому верхньому куті.

The image shows two side-by-side screenshots of the 'Dia App' user profile screen. Both screens have a blue header with 'Dia App' and a status bar at the top showing the time as 12:21 and battery at 66% (left) and 68% (right). Red 'demo' labels are in the top right corners.

Left Screenshot (Successful State):

- Top section: 'Надішліть цей id своєму лікарю:' followed by 'Patient987654321' and a blue button 'Надіслати id'.
- Form fields:
 - 'Ім'я користувача' with the value 'Patient'.
 - 'Електронна пошта лікаря' with the value 'doctor@gmail.com'.
- Bottom: A blue button 'Зберегти'.

Right Screenshot (Error State):

- Top section: Same as the left screenshot.
- Form fields:
 - 'Ім'я користувача' with the value 'Приклад: Іван Іванов'.
 - 'Електронна пошта лікаря' with the value 'doctor'.
- Feedback: A red error message 'Невірний формат електронної пошти' is displayed below the email field.
- Bottom: A blue button 'Зберегти'.

Рисунок 5.21 – Сторінка особистих даних пацієнта

Для відправлення особистого ідентифікатору користувача лікарю, застосунок склав лист для електронної пошти, приклад якого наведено на рисунку 5.22.

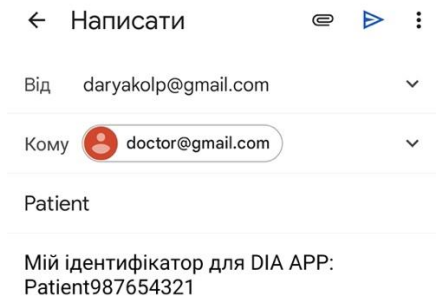


Рисунок 5.22 – Лист лікарю з ідентифікатором пацієнта

Для отримання статистики, необхідно натиснути на кнопку на панелі навігації “Статистика”, де наведено список статистичних даних який має сортування за принципом сторінки діагностики (див. рис. 5.23).

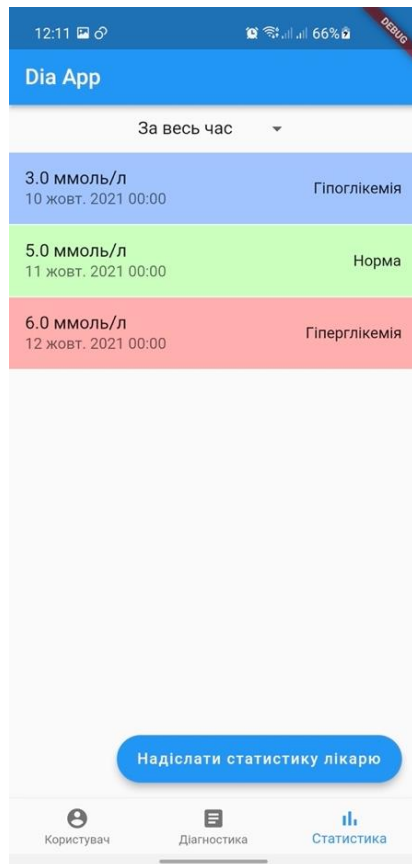


Рисунок 5.23 – Лист лікарю з ідентифікатором пацієнта

Якщо, лікар не зареєстрований у системі, то застосунок має можливість відправити ститистичні дані лікарю електроною поштою, автоматично склавши листа, приклад якого зображено на рисунку 5.24.

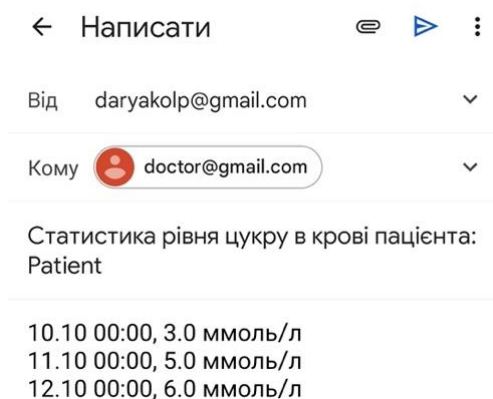


Рисунок 5.24 – Лист лікарю зі статистикою глюкози у крові пацієнта за обраний період

Як було вже зазначено, система має два типи користувачів. Тому як було здійснено авторизацію визначається тип: лікар або пацієнт. На рисунку 5.25 наведено екран авторизації. Як можна побачити авторизація зручна, та не вимагає від користувачів введення логіну та паролю.

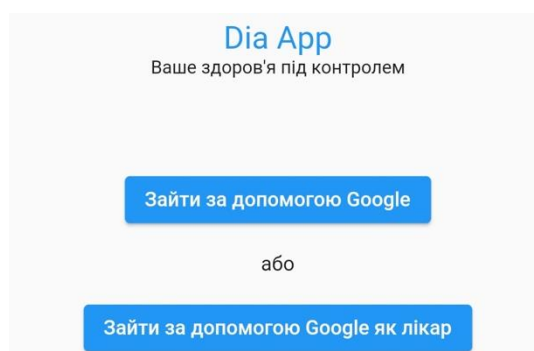


Рисунок 5.25 – Сторінка авторизації

Після того як лікар авторизувався, він має доступ до списку пацієнтів яких спостерігає (див. рис. 5.17). Для додавання нових пацієнтів лікар повинен знати ідентифікатор пацієнта та ввести дані, скориставшись формою для додавання нового пацієнта, яка знаходиться у верхній часті сторінки “Пацієнти”. При натисканні на пункт будь-якого пацієнта, застосунок відкриває сторінку де знаходиться статистика глюкози в крові, яка закріплена за пацієнтом (див. рис. 5.26). На відміну від сторінки “Статистика” в застосунку пацієнта, ця сторінка має інформацію про ім’я пацієнта.

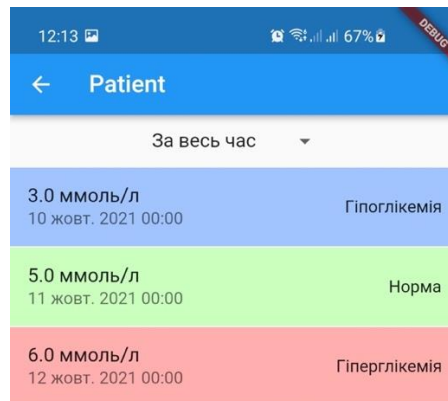


Рисунок 5.26 – Сторінка статистичних даних пацієнта

У лікаря також є сторінка з особистими даними, яку наведено на рисунку 5.27. Як можна побачити, лікар не має можливості виправити ці дані, вони формуються під час реєстрації. Неможливість редагування обумовлена тим, що пацієнти знають про пошту та ім'я лікаря, тому зміна будь-яких даних про нього може призвести до плутанини.

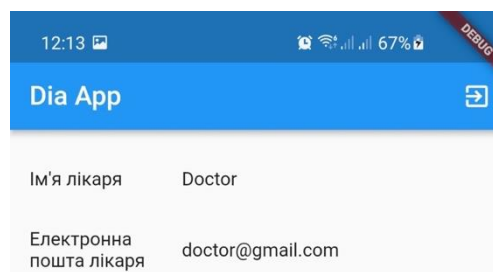


Рисунок 5.27 – Сторінка особистих даних лікаря

Роблячи висновок, можна затвердити, що застосунок відповідає заявленим вимогам та є простим та інтуїтивно зрозумілим. Слід відзначити, що для демонстрації можливостей застосунку було використано реалізацію фіктивних репозиторіїв з префіксом Mock, щоб не витрачати квоти Firebase. Реалізацію та ефективність такого підходу було описано у попередніх розділах. Для цього у main.dart було замінено реалізацію репозиторіїв з префіксом Firebase (див. рис. 4.12) на Mock.

5.4 Модульне тестування програми

У попередньому розділі, було реалізовано тести для впевненості у коректній роботі застосунку, а також надано приклад коду тестів (повну версію коду надано у Додатку А). Для тестування було використано репозиторії з префіксом Mock. Результати тестування можна побачити на рисунках, які наведено далі.

На рисунках 5.28 та 5.29 представлено результати тестування сутностей користувача та статистики, які є основою бізнес-логіки застосунку. Згідно з результатами роботи тестів, очевидно, що код сутностей Patient, Doctor та BloodSugarStatistic, які є ключовими для розробленого застосунку, працює коректно та без збоїв.

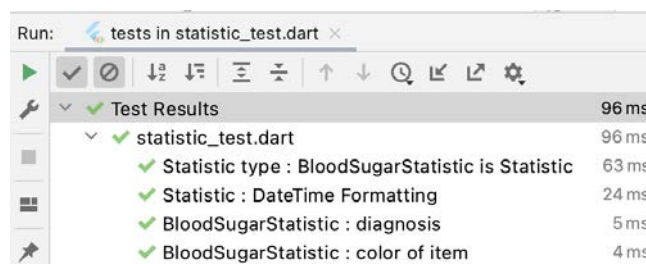


Рисунок 5.28 – Результати тестування Statistic та BloodSugarStatistic сутностей

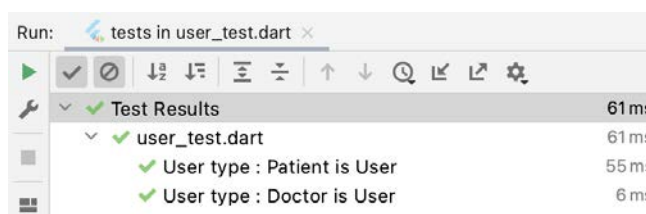


Рисунок 5.29 – Результати тестування моделі користувача

Далі, на рисунку 5.30 показано як відпрацьовує код, який відповідальний за авторизацію, а на рисунку 5.31 – результати тестування репозиторію, який було розроблено для роботи з користувачами. Також було

проведено тестування фабрики, яка відповідальна за створення репозиторію в залежності від того, який користувач авторизувався у системі.

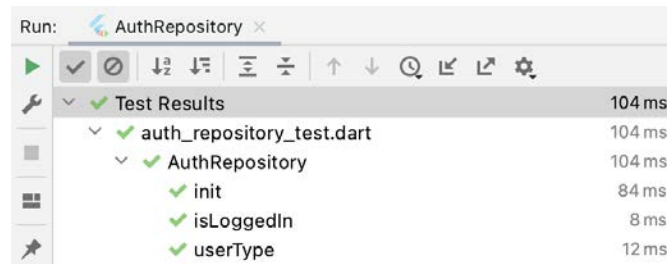


Рисунок 5.30 – Результати тестування репозиторію авторизації

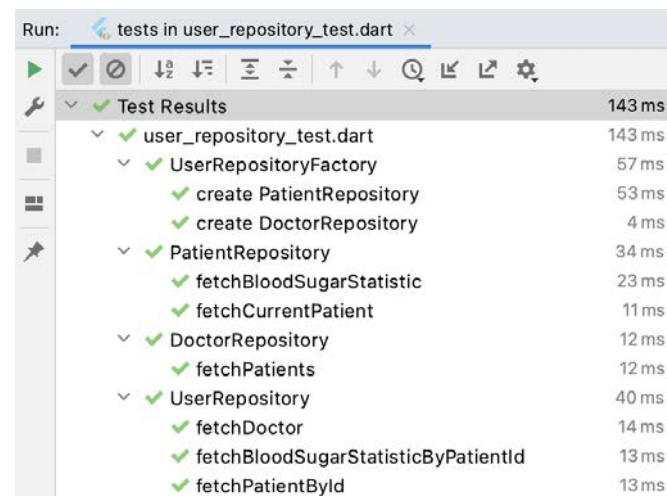


Рисунок 5.31 – Результати тестування репозиторіїв користувача

Необхідність модульного тестування полягає у тому, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку. Згідно проведеним тестам, код репозиторіїв для авторизації та для користувача є коректним та повністю виконує свої функції, які передбачені вимогами до проєкту.

5.5 Висновок до п'ятого розділу

У даному розділі розроблено сценарії роботи клієнтів з Cloud Firestore та, на їх базі, протестовано запити на доступ та безпеку даних, проведено експерименти зчитування, додавання, редагування даних Cloud Firestore. З

метою проведення ручного тестування взаємодії бекенд-сервісу та застосунку, у проєкті застосунку використано репозиторії з префіксом Firebase. Для демонстрації та тестування можливостей інтерфейсу користувача та для проведення модульного тестування було використано реалізацію фіктивних репозиторіїв з префіксом Mock, щоб не витрачати ресурси Firebase. За результатами тестування у програмному продукті помилок та збоїв не виявлено, розроблений програмний продукт виконує свої функції правильно, ефективно та відповідає вимогам.

ВИСНОВКИ

Метою виконання дипломної кваліфікаційної роботи є розробка архітектури застосунку, який взаємодіє з платформою сервісних послуг, що надає доступ до хмарної бази даних. Результатом роботи є застосунок для пацієнтів з цукровим діабетом та лікарів, який використовує Firebase у якості серверу.

Для досягнення поставленої мети у роботі, було вирішено наступні завдання:

а) проаналізовано існуючі бекенд як сервіси: Firebase, AWS Amplify та Backendless, для розробки застосунку було обрано Firebase у зв'язку з тим, що ця платформа надійніша за інші рішення та має більше безкоштовних сервісів. Однак виявлено, що згідно з теоремою CAP, усі з оглянутих сервісів можуть бути орієнтовані на узгодженість, так що усі вони можуть бути використані для впровадження у застосунок.

б) вивчено стан проблеми захворювання на цукровий діабет в Україні та світі. З'ясовано, що кількість захворюваних в Україні за 2020 рік становить 1 млн. 134 тис. чоловік, що означає, що проблема цукрового діабету чутлива для України.

в) розглянуто існуючі програмні рішення для людей хворих на діабет та виявлено, що жоден з розглянутих рішень не надає лікарям можливість отримувати дані своїх пацієнтів напряму через застосунок, лише шляхом отримання листа на електронну пошту.

г) основуючись на дослідженнях предметної галузі, було створено бізнес модель з використанням підходу “Канвас”, яка описує бізнес-процеси та перспективи розвитку розробленого застосунку для людей хворих на діабет.

д) враховуючи принципи чистої архітектури, розроблено детальну схему архітектури застосунку, запропоновано методи впровадження API бекенду як сервісу у проєкт.

е) описано бізнес-логіку програмного продукту. Розроблено архітектуру проєкту, з урахуванням запропонованих методів, створено адаптивний архітектурний шаблон. Детально описано взаємодію класів та компонентів проєкту.

є) обґрунтовано вибір програмного каркасу Flutter для розробки. Розглянуто особливості та вимоги для створення програмного продукту за допомогою Flutter. Описано процес налаштування сервісів Firebase. Детально описано розробку застосунку, з дотриманням принципів розробленої архітектури.

ж) протестовано сценарії використання Cloud Firestore, проведено ручне тестування взаємодії застосунка з платформою, продемонстровано можливості інтерфейсу розробленого застосунку, проведено модульне тестування. За результатами тестування помилок не виявлено.

Розробка застосунку здійснювалась за допомогою об'єктно-орієнтованого програмування з використанням мови програмування Dart та програмного каркасу Flutter та середовища розробки Android Studio. Розроблений застосунок знаходиться за посиланням – [37].

Часткові результати даної кваліфікаційної роботи, було подано як тезу на науково-практичну конференцію викладачів, науковців, молодих учених, аспірантів та студентів “Тиждень науки-2021”.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Rise of Mobile Backend as a Service (MBaaS) API Stacks [Electronic resource]. – Access mode: <https://apievangelist.com/2012/06/03/rise-of-mobile-backend-as-a-service-mbaas-api-stacks/>.
2. Clark, J. Top 10 Mobile App Backend Hosting Providers [Electronic resource] / J. Clark. – Access mode: <https://blog.back4app.com/mobile-app-backend-hosting/>.
3. Geewax, J.J. API Design Patterns [Text] / John J. Geewax. – New York: Simon and Schuster, 2021. – 480 p.
4. Firebase [Electronic resource]. – Access mode: <https://firebase.google.com/>.
5. AWS Amazon [Electronic resource]. – Access mode: <https://aws.amazon.com/ru/amplify/>.
6. Backendless [Electronic resource]. – Access mode: <https://develop.backendless.com/>.
7. Firebase, Pricing plans [Electronic resource]. – Access mode: <https://firebase.google.com/pricing?authuser=0>.
8. Backendless, Pricing plans [Electronic resource]. – Access mode: <https://backendless.com/pricing/backendless-cloud>.
9. Каминский, А.В. Сахарный диабет и ожирение: клиническое руководство по диагностике и лечению [Текст] / А.В. Каминский, А.Н. Коваленко. – К.: Салютис, 2010. – 256 с.
10. International Diabetes Federation, Diabetes facts & figures [Electronic resource]. – Access mode: <https://idf.org/aboutdiabetes/what-is-diabetes/facts-figures.html>.
11. Центр Медичної статистики, Всесвітній день боротьби з цукровим діабетом [Електрон. ресурс]. – Режим доступу: <https://zoiaCMS.zp.ua/2020/11/16/vsesvitnii-den-borotby-z-tsukrovym-diabetom/>.

12. MySugr – Diabetes Tracker Log [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.mysugr.android.companion>.
13. Blood Glucose Tracker [Electronic resource]. – Access mode: <https://play.google.com/store/apps/details?id=mobi.littlebytes.android.bloodglucose.tracker>.
14. OneTouch Reveal – Контроль здоровья при диабете [Электрон. ресурс]. – Режим доступа: <https://play.google.com/store/apps/details?id=ru.onetouchreveal.app>.
15. Судакова, Н. Как построить прибыльную бизнес-модель Канвас: шаблон и примеры [Электрон. ресурс] / Н. Судакова. – Режим доступа: pravda.com.ua/rus/publications/2020/08/1/663610/.
16. CAP and PACELC theorems in plain English [Electronic resource]. – Access mode: <https://luminousmen.com/post/cap-and-pacelc-theorems-in-plain-english/>.
17. Johnson, J. CAP Theorem for Databases: Consistency, Availability & Partition Tolerance [Electronic resource] / J. Johnson. – Access mode: <https://www.bmc.com/blogs/cap-theorem/>.
18. Мартин, Р. Чистая архитектура, искусство разработки программного обеспечения [Текст] / Р. Мартин. – СПб: Прогресс книга, 2018. – 352 с.
19. pub.dev, Cloud Firestore Plugin for Flutter [Electronic resource]. – Access mode: https://pub.dev/packages/cloud_firestore.
20. pub.dev, Backendless SDK for Flutter [Electronic resource]. – Access mode: https://pub.dev/packages/backendless_sdk.
21. pub.dev, Amplify Plugin for Flutter [Electronic resource]. – Access mode: https://pub.dev/packages/amplify_flutter.
22. ElHady, H. Flutter vs. React Native vs. Xamarin [Electronic resource] / H. ElHady. – Access mode: <https://blog.logrocket.com/flutter-vs-react-native-vs-xamarin/>.

23. Feoktistov, I. Top 8 Flutter Advantages and Why You Should Try Flutter on Your Next Project [Electronic resource] / I. Feoktistov. – Access mode: <https://relevant.software/blog/top-8-flutter-advantages-and-why-you-should-try-flutter-on-your-next-project/>.

24. Leler, W. What's Revolutionary about Flutter? [Electronic resource] / W. Leler. – Access mode: <https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514>.

25. Clow, M. Learn Google Flutter Fast: 65 Example Apps [Text] / M. Clow. – London: Independently Published, 2019. – 507 p.

26. Дикяк В. Стоит ли инвестировать во Flutter. Сравнение Flutter и React Native [Электрон. ресурс] / В. Дикяк. – Режим доступа: <https://dou.ua/lenta/articles/flutter-for-mobile-apps/>.

27. Jasani, M. React Native vs Flutter vs Xamarin: A Comparative Guide [2020] [Electronic resource] / M. Jasani. – Access mode: <https://www.codementor.io/@mahil/react-native-vs-flutter-vs-xamarin-a-comparative-guide-2020-11evx1f7q4>.

28. Widget class [Electronic resource]. – Access mode: <https://api.flutter.dev/flutter/widgets/Widget-class.html>.

29. List of state management approaches [Electronic resource]. – Access mode: <https://flutter.dev/docs/development/data-and-backend/state-mgmt/options>.

30. BuildContext class [Electronic resource]. – Access mode: <https://api.flutter.dev/flutter/widgets/BuildContext-class.html>.

31. pub.dev, GetIt [Electronic resource]. – Access mode: https://pub.dev/packages/get_it.

32. pub.dev, Injectable [Electronic resource]. – Access mode: <https://pub.dev/packages/injectable>.

33. Dart Documentation [Electronic resource]. – Access mode: <https://dart.dev/guides>.

34. Android Studio vs Visual Studio Code: What are the differences? [Electronic resource]. – Access mode: <https://stackshare.io/stackups/android-studio->

vs-visual-studio-code.

35. Google Pay Services, Authenticating Your Client [Electronic resource].
– Access mode: <https://developers.google.com/android/guides/client-auth>

36. Firebase, Basic Security Rules [Electronic resource]. – Access mode:
<https://firebase.google.com/docs/rules/basics?authuser=0>.

37. GitHub: сторінка розробленого ПЗ [Електрон.ресурс]. – Режим
доступу: https://github.com/dina200/dia_app.

ДОДАТОК А
ТЕКСТ ПРОГРАМИ

A.1 Файл запуска програми lib/main.dart

```

import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/dia_localizations.dart';
import 'package:get_it/get_it.dart';
import 'src/app/pages/login/login_page.dart';
import 'src/app/pages/main/main_page.dart';
import 'src/app/pages/main_doctor/main_doctor_page.dart';
import 'src/data/firebase_repositories/firebase_auth_repository.dart';
import 'src/data/firebase_repositories/firebase_user_repository.dart';
import 'src/device/utils/google_service.dart';
import 'src/device/utils/store_interactor.dart';
import 'src/domain/entities/user.dart';
import 'src/domain/repositories_contracts/auth_repository.dart';
import 'src/domain/repositories_contracts/user_repository.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  GetIt.I
    ..registerSingleton<GoogleService>(GoogleService())
    ..registerSingleton<StoreInteractor>(StoreInteractor())
    ..registerSingleton<AuthRepository>(FirebaseAuthRepository())
    ..registerSingleton<UserRepositoryFactory>(FirebaseUserRepositoryFactory());
  final userType = await GetIt.I<AuthRepository>().init();
  runApp(MyApp(userType: userType));
}

class MyApp extends StatelessWidget {
  final UserType userType;
  const MyApp({
    Key key,
    @required this.userType,
  }) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      onGenerateTitle: _onGenerateTitle,
      onGenerateRoute: _onGenerateRoute,
      localizationsDelegates: DiaLocalizations.localizationsDelegates,
      supportedLocales: DiaLocalizations.supportedLocales,
    );
  }
  String _onGenerateTitle(BuildContext context) {
    return DiaLocalizations.of(context).appName;
  }
  Route _onGenerateRoute(RouteSettings settings) {
    if (userType != null) {
      switch (userType) {
        case UserType.Patient: return MainPage.buildPageRoute();
        case UserType.Doctor: return MainDoctorPage.buildPageRoute();
      }
    }
    return LoginPage.buildPageRoute();
  }
}

```

A.2 Код моделі користувача lib/src/domain/entities/user.dart

```
import 'package:meta/meta.dart';

enum UserType {Patient, Doctor}
abstract class User {
  final String id;
  final String email;
  final String fullName;
  User({
    @required this.id,
    @required this.email,
    this.fullName,
  }) : assert(id != null), assert(email != null);
  @override
  bool operator ==(Object other) =>
    identical(this, other) ||
    other is User &&
    runtimeType == other.runtimeType &&
    id == other.id &&
    email == other.email &&
    fullName == other.fullName;
  @override
  int get hashCode => id.hashCode ^ email.hashCode ^ fullName.hashCode;
}
class Patient extends User {
  final String docEmail;
  Patient({String id, String email, String fullName, this.docEmail})
    : super (id: id, email: email, fullName: fullName);
}
class Doctor extends User {
  final List<String> patientIds;
  Doctor({String id, String email, String fullName, this.patientIds})
    : super (id: id, email: email, fullName: fullName);
}
```

A.3 Код моделі статистики lib/src/domain/entities/statistic.dart

```
import 'dart:ui';
import 'package:meta/meta.dart';
import 'package:intl/intl.dart';
import 'package:flutter_gen/gen_l10n/dia_localizations.dart';

abstract class Statistic {
  final DateTime dateTimeOfMeasure;
  Statistic({DateTime dateTimeOfMeasure})
    : dateTimeOfMeasure = dateTimeOfMeasure ?? DateTime.now();
  int compareTo(Statistic anotherStatistic) {
    if (dateTimeOfMeasure.isAfter(anotherStatistic.dateTimeOfMeasure)) return -1;
    else if (dateTimeOfMeasure.isBefore(anotherStatistic.dateTimeOfMeasure)) return 1;
    else return 0;
  }
  String getDate(DiaLocalizations locale) {
    final date = DateFormat.yMMMd(locale.localeName).format(dateTimeOfMeasure);
    final time = DateFormat.Hm(locale.localeName).format(dateTimeOfMeasure);
    return '$date $time';
  }
  String getDate(DiaLocalizations locale) {
    return DateFormat.yMd(locale.localeName).format(dateTimeOfMeasure);
  }
  String getSimpleDateTime() {
    return DateFormat('dd.MM HH:mm').format(dateTimeOfMeasure);
  }
}
```

```

    }
}
class BloodSugarStatistic extends Statistic {
    final double bloodSugar;
    BloodSugarStatistic({
        @required this.bloodSugar,
        DateTime dateTimeOfMeasure,
    }) : assert(bloodSugar != null),
        super(dateTimeOfMeasure: dateTimeOfMeasure);
    BloodSugarDiagnosis get diagnosis {
        if (bloodSugar < 4.1) {
            return BloodSugarDiagnosis.hypoglycemia;
        } else if (bloodSugar > 5.9) {
            return BloodSugarDiagnosis.hyperglycemia;
        }
        return BloodSugarDiagnosis.normal;
    }
    String getFullData(DiaLocalizations locale) {
        final formattedDateTime = getDateTime(locale);
        final dataOfBloodSugar = getBloodSugar(locale);
        final bloodSugarDiagnosis = getDiagnosis(locale);
        return '$formattedDateTime, $dataOfBloodSugar, $bloodSugarDiagnosis';
    }
    String getDiagnosis(DiaLocalizations locale) {
        switch (diagnosis) {
            case BloodSugarDiagnosis.hypoglycemia: return locale.hypoglycemia;
            case BloodSugarDiagnosis.hyperglycemia: return locale.hyperglycemia;
            case BloodSugarDiagnosis.normal: return locale.normal;
            default: return '-';
        }
    }
    String getBloodSugar(DiaLocalizations locale) {
        final dataOfBloodSugar = '${bloodSugar.toStringAsFixed(1).toString()}';
        return '$dataOfBloodSugar ${locale.mmolPerL}';
    }
    Color get color {
        switch (diagnosis) {
            case BloodSugarDiagnosis.hypoglycemia: return Color(0xFFA0C4FF);
            case BloodSugarDiagnosis.normal: return Color(0xFFCAFFBF);
            case BloodSugarDiagnosis.hyperglycemia: return Color(0xFFffadad);
            default: return Color(0xFFFFFFFF);
        }
    }
    String getBriefInfo(DiaLocalizations locale) {
        return '${super.getSimpleDateTime()}, ${getBloodSugar(locale)}';
    }
    @override
    bool operator ==(Object other) =>
        identical(this, other) ||
        other is BloodSugarStatistic &&
        runtimeType == other.runtimeType &&
        dateTimeOfMeasure == other.dateTimeOfMeasure &&
        bloodSugar == other.bloodSugar;
    @override
    int get hashCode => bloodSugar.hashCode;
}
enum BloodSugarDiagnosis {
    hypoglycemia,
    hyperglycemia,
    normal,
}

```

A.4 Код абстрактного класу AuthRepository

lib/src/domain/repositories_contracts/auth_repository.dart

```
import 'package:flutter/foundation.dart';
import 'package:get_it/get_it.dart';
import 'package:dia_app/src/domain/entities/user.dart';
import 'package:dia_app/src/domain/repositories_contracts/user_repository.dart';

abstract class AuthRepository {
  Future<bool> get isLoggedIn;
  Future<UserType> get userType;
  Future<UserType> init() async {
    if (await isLoggedIn) {
      final type = await userType;
      _registerUserRepository(type);
      return type;
    }
    return null;
  }
  @mustCallSuper
  Future<void> loginWithGoogleAsPatient() async {
    if (await isLoggedIn) {
      _registerUserRepository(UserType.Patient);
    }
  }
  @mustCallSuper
  Future<void> loginWithGoogleAsDoctor() async {
    if (await isLoggedIn) {
      _registerUserRepository(UserType.Doctor);
    }
  }
  @mustCallSuper
  Future<void> logOut() async {
    GetIt.I.unregister<UserRepository>();
  }
  void _registerUserRepository(UserType userType) {
    GetIt.I.registerSingleton<UserRepository>(
      GetIt.I.<UserRepositoryFactory>().createUserRepository(userType),
    );
  }
}
```

A.5 Код абстрактного класу UserRepository

lib/src/domain/repositories_contracts/user_repository.dart

```
import 'package:dia_app/src/domain/entities/statistic.dart';
import 'package:dia_app/src/domain/entities/time_range.dart';
import 'package:dia_app/src/domain/entities/user.dart';

abstract class UserRepositoryFactory {
  UserRepository createUserRepository(UserType type);
}

abstract class UserRepository {
  Future<Doctor> fetchDoctor();
  Future<Patient> fetchPatientById(String patientId);
  Future<List<BloodSugarStatistic>> fetchBloodSugarStatisticByPatientId(
    String patientId, [
      TimeRange timeRange,
    ]);
}
```

```

}
abstract class PatientRepository extends UserRepository {
  Future<void> changePatientData(String name, String docEmail);
  Future<Patient> fetchCurrentPatient();
  Future<void> saveBloodSugarData(BloodSugarStatistic data);
  Future<List<BloodSugarStatistic>> fetchBloodSugarStatistic([
    TimeRange timeRange,
  ]);
}
abstract class DoctorRepository extends UserRepository {
  Future<void> addPatient(String patientId);
  Future<List<Patient>> fetchPatients();
}

```

A.6

Код

реалізації

FirebaseAuthRepository

lib/src/data/firebase_repositories/firebase_auth_repository.dart

```

import 'package:firebase_auth/firebase_auth.dart' as auth;
import 'package:get_it/get_it.dart';
import 'package:dia_app/src/data/firebase_repositories/dia_firestore_helper.dart';
import 'package:dia_app/src/domain/entities/user.dart';
import 'package:dia_app/src/device/utils/google_service.dart';
import 'package:dia_app/src/device/utils/store_interactor.dart';
import 'package:dia_app/src/domain/repositories_contracts/auth_repository.dart';
import 'dia_firestore_helper.dart';

```

```

class FirebaseAuthRepository extends AuthRepository {
  final auth.FirebaseAuth _firebaseAuth;
  final GoogleService _googleService;
  final StoreInteractor _storeInteractor;
  FirebaseAuthRepository()
    : _firebaseAuth = auth.FirebaseAuth.instance,
      _googleService = GetIt.I<GoogleService>(),
      _storeInteractor = GetIt.I<StoreInteractor>();
  @override
  Future<bool> get isLoggedIn async {
    final currentUser = _firebaseAuth.currentUser;
    final token = await _storeInteractor.getToken();
    return currentUser != null && token != null;
  }
  @override
  Future<UserType> get userType async {
    if (await isLoggedIn) {
      return await _storeInteractor.getUserType();
    }
    return null;
  }
  @override
  Future<void> loginWithGoogleAsPatient() async {
    final googleAuthentication = await _googleService.getGoogleAuthData();
    final credential = auth.GoogleAuthProvider.credential(
      idToken: googleAuthentication.idToken,
      accessToken: googleAuthentication.accessToken,
    );
    final authResult = await _firebaseAuth.signInWithCredential(credential);
    await _savePatientData(authResult.user);
    super.loginWithGoogleAsPatient();
  }
}

```

@override

```

Future<void> loginWithGoogleAsDoctor() async {
  final googleAuthentication = await _googleService.getGoogleAuthData();
  final credential = auth.GoogleAuthProvider.credential(
    idToken: googleAuthentication.idToken,
    accessToken: googleAuthentication.accessToken,
  );
  final authResult = await _firebaseAuth.signInWithCredential(credential);
  await _saveDoctorData(authResult.user);
  super.loginWithGoogleAsDoctor();
}
@override
Future<void> logout() async {
  await _googleService.signOut();
  await _storeInteractor.removeType();
  await _storeInteractor.removeToken();
  super.logout();
}
Future<void> _savePatientData(auth.User user) async {
  await _storeInteractor.setToken(user.uid);
  await _storeInteractor.setUserType(UserType.Patient);
  await DiaFirestoreHelper.savePatientData(
    await _storeInteractor.getToken(),
    user.displayName,
    user.email,
  );
}
Future<void> _saveDoctorData(auth.User user) async {
  await _storeInteractor.setToken(user.uid);
  await _storeInteractor.setUserType(UserType.Doctor);
  await DiaFirestoreHelper.saveDoctorData(
    await _storeInteractor.getToken(),
    user.displayName,
    user.email,
  );
}
}
}

```

A.7 Код допоміжного класу GoogleService

lib/src/device/utils/google_service.dart

```

import 'dart:async';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:dia_app/src/domain/entities/exceptions.dart';

class GoogleService {
  final _googleSignIn = GoogleSignIn(scopes: <String>['email']);
  Future<GoogleSignInAuthentication> getGoogleAuthData() async {
    final account = await _googleSignIn.signIn();
    final isSignedIn = await _googleSignIn.isSignedIn();
    if (isSignedIn) {
      try {
        return await account.authentication;
      } finally {
        await _googleSignIn.disconnect();
      }
    }
    throw GoogleLoginException();
  }
  Future<void> signOut() async {
    await _googleSignIn.signOut();
  }
}

```

```

}
}

```

A.8 Код фіктивної реалізації MockAuthRepository

lib/src/data/mock_repositories/mock_auth_repository.dart

```

import 'package:dia_app/src/domain/entities/user.dart';
import 'package:dia_app/src/domain/repositories_contracts/auth_repository.dart';

class MockAuthRepository extends AuthRepository {
  @override
  Future<bool> get isLoggedIn async {
    return true;
  }
  @override
  Future<UserType> get userType async => UserType.Patient;
  @override
  Future<void> loginWithGoogleAsPatient() async {
    super.loginWithGoogleAsPatient();
  }
  @override
  Future<void> loginWithGoogleAsDoctor() async {
    super.loginWithGoogleAsDoctor();
  }
  @override
  Future<void> logOut() async {
    super.logOut();
  }
}

```

A.9 Код реалізації FirebaseUserRepository

lib/src/data/firebase_repositories/firebase_user_repository.dart

```

import 'dart:async';
import 'package:get_it/get_it.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/foundation.dart';
import 'package:dia_app/src/data/firebase_repositories/dia_firestore_helper.dart';
import 'package:dia_app/src/device/utills/store_interactor.dart';
import 'package:dia_app/src/domain/entities/statistic.dart';
import 'package:dia_app/src/domain/entities/time_range.dart';
import 'package:dia_app/src/domain/entities/user.dart';
import 'package:dia_app/src/domain/repositories_contracts/user_repository.dart';

class FirebaseUserRepositoryFactory extends UserRepositoryFactory {
  @override
  UserRepository createUserRepository(UserType type) {
    switch(type) {
      case UserType.Patient:
        return FirebasePatientRepository();
      case UserType.Doctor:
        return FirebaseDoctorRepository();
      default:
        throw AssertionError("This user type do not exist");
    }
  }
}

```

```

class FirebasePatientRepository extends _FirebaseUserRepository
    implements PatientRepository {
    @override
    Future<void> changePatientData(String name, String docEmail) async {
        await DiaFirestoreHelper.changePatientData(await _token, name, docEmail);
    }
    @override
    Future<Patient> fetchCurrentPatient() async {
        return await fetchPatientById(await _token);
    }

    @override
    Future<void> saveBloodSugarData(BloodSugarStatistic data) async {
        await DiaFirestoreHelper.setBloodSugarData(
            await _token,
            data.dateTimeOfMeasure,
            data.bloodSugar,
        );
    }
    @override
    Future<List<BloodSugarStatistic>> fetchBloodSugarStatistic([
        TimeRange timeRange,
    ]) async {
        return await fetchBloodSugarStatisticByPatientId(await _token, timeRange);
    }
}

class FirebaseDoctorRepository extends _FirebaseUserRepository
    implements DoctorRepository {
    @override
    Future<void> addPatient(String patientId) async {
        final data = await DiaFirestoreHelper.fetchPatientData(patientId);
        if (data == null) {
            throw AssertionError();
        }
        await DiaFirestoreHelper.addPatient(await _token, patientId);
    }
    @override
    Future<List<Patient>> fetchPatients() async {
        final doctor = await fetchDoctor();
        if (doctor.patientIds != null) {
            final mappedList = doctor.patientIds.map(
                (id) async {
                    return await fetchPatientById(id);
                },
            );
            return await Future.wait(mappedList);
        }
        return [];
    }
}

abstract class _FirebaseUserRepository extends UserRepository {
    Future<String> get _token async {
        return await GetIt.I<StoreInteractor>().getToken();
    }
    @override
    Future<Doctor> fetchDoctor() async {
        final data = await DiaFirestoreHelper.fetchDoctorData(await _token);
        return Doctor(
            id: data[ID],
            email: data[EMAIL],
            fullName: data[NAME],
            patientIds: data[PATIENTS],
        );
    }
}

```

```

@override
Future<Patient> fetchPatientById(String patientId) async {
  final data = await DiaFirestoreHelper.fetchPatientData(patientId);
  return Patient(
    id: data[ID],
    email: data[EMAIL],
    fullName: data[NAME],
    docEmail: data[DOC_EMAIL],
  );
}
@override
Future<List<BloodSugarStatistic>> fetchBloodSugarStatisticByPatientId(
  String patientId, [
    TimeRange timeRange,
  ]) async {
  if (timeRange == null) {
    timeRange = TimeRange.getEntityByFilter(TimeRangeFilters.wholeTime);
  }
  final data = await DiaFirestoreHelper.fetchBloodSugarStatistic(
    patientId,
    timeRange.from,
    timeRange.to,
  );
  return await compute(_parseStatistic, data);
}
FutureOr<List<BloodSugarStatistic>> _parseStatistic(
  List<QueryDocumentSnapshot> data,
) {
  return data.map((s) {
    return BloodSugarStatistic(
      bloodSugar: s[BLOOD_SUGAR],
      dateTimeOfMeasure: (s[TIME_OF_MEASURE] as Timestamp).toDate(),
    );
  }).toList();
}

```

A.10 Код фіктивної реалізації MockUserRepository

lib/src/data/mock_repositories/mock_user_repository.dart

```

import 'dart:async';
import 'package:dia_app/src/domain/entities/statistic.dart';
import 'package:dia_app/src/domain/entities/time_range.dart';
import 'package:dia_app/src/domain/entities/user.dart';
import 'package:dia_app/src/domain/repositories_contracts/user_repository.dart';

class MockUserRepositoryFactory implements UserRepositoryFactory {
  @override
  UserRepository createUserRepository(UserType type) {
    switch(type) {
      case UserType.Patient:
        return MockPatientRepository();
      case UserType.Doctor:
        return MockDoctorRepository();
      default:
        throw AssertionError('This user type do not exist!');
    }
  }
}

class MockPatientRepository extends _MockUserRepository

```

```

    implements PatientRepository {
    @override
    Future<void> changePatientData(String name, String docEmail) async {}

    @override
    Future<List<BloodSugarStatistic>> fetchBloodSugarStatistic([
        TimeRange timeRange,
    ]) async {
        return fetchBloodSugarStatisticByPatientId('Patient987654321');
    }
    @override
    Future<Patient> fetchCurrentPatient() async {
        return fetchPatientById('Patient987654321');
    }
    @override
    Future<void> saveBloodSugarData(BloodSugarStatistic data) async {}
}
class MockDoctorRepository extends _MockUserRepository
    implements DoctorRepository {
    @override
    Future<void> addPatient(String patientId) async {}
    @override
    Future<List<Patient>> fetchPatients() async {
        return [
            Patient(
                id: 'Patient987654321',
                fullName: 'Patient',
                email: 'patient@gmail.com',
                docEmail: 'doctor@gmail.com',
            ),
            Patient(
                id: 'Patient9876543211',
                fullName: 'Patient1',
                email: 'patient1@gmail.com',
                docEmail: 'doctor@gmail.com',
            ),
            Patient(
                id: 'Patient9876543212',
                fullName: 'Patient2',
                email: 'patient2@gmail.com',
                docEmail: 'doctor@gmail.com',
            ),
            Patient(
                id: 'Patient9876543213',
                fullName: 'Patient3',
                email: 'patient3@gmail.com',
                docEmail: 'doctor@gmail.com',
            ),
        ];
    }
}
abstract class _MockUserRepository implements UserRepository {
    @override
    Future<List<BloodSugarStatistic>> fetchBloodSugarStatisticByPatientId(
        String patientId,
        [TimeRange timeRange]) async {
        return [
            BloodSugarStatistic(
                bloodSugar: 3.0,
                dateTimeOfMeasure: DateTime(2021, 10, 10),
            ),
            BloodSugarStatistic(
                bloodSugar: 5.0,

```

```

        dateTimeOfMeasure: DateTime(2021, 10, 11),
      ),
      BloodSugarStatistic(
        bloodSugar: 6.0,
        dateTimeOfMeasure: DateTime(2021, 10, 12),
      ),
    ],
  },
  @override
Future<Doctor> fetchDoctor() async {
  return Doctor(
    id: 'Doctor987654321',
    fullName: 'Doctor',
    email: 'doctor@gmail.com',
    patientIds: [
      'Patient987654321',
    ],
  );
}
@override
Future<Patient> fetchPatientById(String patientId) async {
  return Patient(
    id: 'Patient987654321',
    fullName: 'Patient',
    email: 'patient@gmail.com',
    docEmail: 'doctor@gmail.com',
  );
}
}
}

```

A.11 Допоміжний клас для взаємодії з CloudFirestore

DiaFirestoreHelper lib/src/data/firebase_repositories/firebase_user_repository.dart

```

import 'package:cloud_firestore/cloud_firestore.dart';

const String PATIENTS = 'patients';
const String DOCTORS = 'doctors';
const String ID = 'id';
const String NAME = 'name';
const String EMAIL = 'email';
const String DOC_EMAIL = 'docEmail';
const String BLOOD_SUGAR_STATISTIC = 'bloodSugarStatistic';
const String TIME_OF_MEASURE = 'timeOfMeasure';
const String BLOOD_SUGAR = 'bloodSugar';

class DiaFirestoreHelper {
  static final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  static Future<void> savePatientData(
    String userId,
    String name,
    String email,
  ) async {
    final userDoc = _getPatientDocRef(userId);
    final userData = await userDoc.get();
    if (!userData.exists) {
      await userDoc.set({ID: userId, NAME: name, EMAIL: email});
    }
  }
}

```

```

static Future<void> saveDoctorData(
    String userId,
    String name,
    String email,
) async {
    final userDoc = _getDoctorDocRef(userId);
    final userData = await userDoc.get();
    if (!userData.exists) {
        await userDoc.set( {ID: userId, NAME: name, EMAIL: email});
    }
}

static Future<void> changePatientData(
    String userId,
    String name,
    String docEmail,
) async {
    await _getPatientDocRef(userId).update( {NAME: name, DOC_EMAIL: docEmail});
}

static Future<Map<String, dynamic>> fetchPatientData(String userId) async {
    final snapshot = await _getPatientDocRef(userId).get();
    return snapshot.data();
}

static Future<Map<String, dynamic>> fetchDoctorData(String userId) async {
    final snapshot = await _getDoctorDocRef(userId).get();
    return snapshot.data();
}

static Future<void> addPatient(String doctorId, String patientId) async {
    await _getDoctorDocRef(doctorId).update( {PATIENTS: FieldValue.arrayUnion([patientId])});
}

static Future<List<QueryDocumentSnapshot>> fetchBloodSugarStatistic(
    String userId,
    DateTime from,
    DateTime to,
) async {
    final snapshot = await _getBloodSugarStatsCollectionRef(userId)
        .where(
            TIME_OF_MEASURE,
            isGreaterThan: Timestamp.fromDate(from),
            isLessThan: Timestamp.fromDate(to),
        )
        .orderBy(TIME_OF_MEASURE, descending: true)
        .get();
    return snapshot.docs;
}

static Future<void> setBloodSugarData(
    String userId,
    DateTime time,
    double bloodSugar,
) async {
    final statisticDoc = _getBloodSugarStatsCollectionRef(userId)
        .doc('${time.millisecondsSinceEpoch}');
    await statisticDoc.set( {
        TIME_OF_MEASURE: Timestamp.fromDate(time),
        BLOOD_SUGAR: bloodSugar,
    });
}

static DocumentReference _getPatientDocRef(String userId) {
    return _firebase.collection(PATIENTS).doc(userId);
}

static DocumentReference _getDoctorDocRef(String userId) {
    return _firebase.collection(DOCTORS).doc(userId);
}

```

```

static CollectionReference _getBloodSugarStatsCollectionRef(String userId) {
  return _getPatientDocRef(userId).collection(BLOOD_SUGAR_STATISTIC);
}
}

```

A.12 Код головної сторінки пацієнта MainPage

lib/src/app/pages/main/main_page.dart

```

import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/dia_localizations.dart';
import 'package:provider/provider.dart';
import 'package:dia_app/src/app/routes.dart' as routes;
import 'package:dia_app/src/app/pages/login/login_page.dart';
import 'package:dia_app/src/app/widgets/loading_layout.dart';
import 'main_page_presenter.dart';
import 'tabs/diagnostics/diagnostics_tab.dart';
import 'tabs/statistics/statistics_tab.dart';
import 'tabs/patient/patient_tab.dart';

class MainPage extends StatefulWidget {
  static const nameRoute = routes.main;
  static PageRoute buildPageRoute() {
    return MaterialPageRoute(
      settings: RouteSettings(name: nameRoute),
      builder: _builder,
    );
  }
  static Widget _builder(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => MainPagePresenter(context),
      child: MainPage(),
    );
  }
  @override
  _MainPageState createState() => _MainPageState();
}

class _MainPageState extends State<MainPage> {
  int _selectedIndex = 1;
  DiaLocalizations get _locale => DiaLocalizations.of(context);
  MainPagePresenter get _watch => context.watch<MainPagePresenter>();
  MainPagePresenter get _read => context.read<MainPagePresenter>();
  @override
  Widget build(BuildContext context) {
    return LoadingLayout(
      isLoading: _watch.isLogOutLoading,
      child: GestureDetector(
        onTap: _resetFocusNode,
        child: Scaffold(
          appBar: AppBar(
            title: Text(_locale.appName),
            actions: [_buildAppBarAction()],
          ),
          bottomNavigationBar: BottomNavigationBar(
            currentIndex: _selectedIndex,
            onTap: _onItemTapped,
            items: <BottomNavigationBarItem>[
              BottomNavigationBarItem(
                icon: Icon(Icons.account_circle),
                label: _locale.user,
              ),
              BottomNavigationBarItem(

```

```

        icon: Icon(Icons.article),
        label: _locale.diagnostics,
      ),
      BottomNavigationBarItem(
        icon: Icon(Icons.bar_chart),
        label: _locale.statistics,
      ),
    ],
  ),
  body: _tabBuilder(),
  floatingActionButton: _buildFloatActionButton(),
),
),
);
}
void _resetFocusNode() {
  FocusScope.of(context).requestFocus(FocusNode());
}
Widget _buildAppBarAction() {
  switch (_selectedIndex) {
    case 0:
      return IconButton(
        icon: Icon(Icons.exit_to_app),
        tooltip: _locale.changeUser,
        onPressed: _logout,
      );
    default: return Container();
  }
}
Future<void> _logout() async {
  await _read.logout();
  await Navigator.of(context).pushAndRemoveUntil(
    LoginPage.buildPageRoute(),
    (route) => false,
  );
}
void _onItemTapped(int index) {
  setState(() => _selectedIndex = index);
}
Widget _tabBuilder() {
  switch (_selectedIndex) {
    case 0: return PatientTab();
    case 1: return DiagnosticsTab();
    case 2: return StatisticsTab();
    default: return Container();
  }
}
Widget _buildFloatActionButton() {
  switch (_selectedIndex) {
    case 2:
      return FloatingActionButton.extended(
        label: Text(_locale.sendDataToDoc),
        onPressed: _watch.isStatisticNotEmpty ? _sendStatistic : null,
      );
    default: return null;
  }
}
Future<void> _sendStatistic() async {
  await _read.sendStatistic();
}
}

```

A.13 Код сценарію використання для головної сторінки пацієнта

MainPagePresenter lib/src/app/pages/main/main_page_presenter.dart

```
import 'dart:async';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/dia_localizations.dart';
import 'package:get_it/get_it.dart';
import 'package:url_launcher/url_launcher.dart';
import 'package:dia_app/src/domain/entities/user.dart';
import 'package:dia_app/src/domain/entities/statistic.dart';
import 'package:dia_app/src/domain/entities/time_range.dart';
import 'package:dia_app/src/domain/repositories_contracts/auth_repository.dart';
import 'package:dia_app/src/domain/repositories_contracts/user_repository.dart';

class MainPagePresenter with ChangeNotifier {
  final BuildContext _context;
  final AuthRepository _authRepository;
  final PatientRepository _patientRepository;

  Patient _patient;
  List<BloodSugarStatistic> _bloodSugarStatistic;
  TimeRangeFilters _timeRangeFilter = TimeRangeFilters.wholeTime;
  bool _isPatientLoading = false;
  bool _isLogOutLoading = false;
  bool _isSavingData = false;
  bool _isStatisticLoading = false;

  Patient get patient => _patient;
  List<BloodSugarStatistic> get bloodSugarStatistic => _bloodSugarStatistic;
  TimeRangeFilters get timeRangeFilter => _timeRangeFilter;
  bool get isUserLoading => _isPatientLoading;
  bool get isLogOutLoading => _isLogOutLoading;
  bool get isSavingData => _isSavingData;
  bool get isStatisticLoading => _isStatisticLoading;
  bool get isStatisticNotEmpty =>
    _bloodSugarStatistic != null && _bloodSugarStatistic.isNotEmpty;

  MainPagePresenter(this._context)
    : _authRepository = GetIt.I.get<AuthRepository>(),
      _patientRepository = GetIt.I.get<UserRepository>() {
    init();
  }

  Future<void> init() async {
    _isPatientLoading = true;
    _isStatisticLoading = true;
    notifyListeners();
    final results = await Future.wait([
      _patientRepository.fetchCurrentPatient(),
      _patientRepository.fetchBloodSugarStatistic(),
    ]);
    _patient = results[0];
    _bloodSugarStatistic = results[1];
    _isPatientLoading = false;
    _isStatisticLoading = false;
    notifyListeners();
  }

  Future<void> saveUserData(String name, String docEmail) async {
    _isPatientLoading = true;
    notifyListeners();
    await _patientRepository.changePatientData(name, docEmail);
    _patient = await _patientRepository.fetchCurrentPatient();
  }
}
```

```

    _isPatientLoading = false;
    notifyListeners();
  }
  Future<void> logOut() async {
    _isLogOutLoading = true;
    notifyListeners();
    await _authRepository.logOut();
    _isLogOutLoading = false;
    notifyListeners();
  }
  Future<void> saveBloodSugarData(String value) async {
    _isSavingData = true;
    notifyListeners();
    await _patientRepository.saveBloodSugarData(
      BloodSugarStatistic(bloodSugar: double.parse(value)),
    );
    _isSavingData = false;
    notifyListeners();
    await sortStatisticByTimeFilter(timeRangeFilter);
    notifyListeners();
  }
  Future<void> sortStatisticByTimeFilter(TimeRangeFilters filter) async {
    _isStatisticLoading = true;
    notifyListeners();
    _timeRangeFilter = filter;
    _bloodSugarStatistic = await _patientRepository
      .fetchBloodSugarStatistic(TimeRange.getEntityByFilter(filter));
    _isStatisticLoading = false;
    notifyListeners();
  }
  Future<void> sendStatistic() async {
    final locale = DiaLocalizations.of(_context);
    final statistic = await compute(_getStatisticToStr, {
      'locale': locale,
      'statistic': _bloodSugarStatistic,
    });
    await launch(
      'mailto:${_patient.docEmail ?? ""}?subject=${locale.bloodSugarStatistic}:
${patient.fullName}&body=$statistic');
  }
  Future<void> sendUserId() async {
    final locale = DiaLocalizations.of(_context);
    await launch(
      'mailto:${_patient.docEmail ?? ""}?subject=${patient.fullName}&body=${locale.sendIdText} ${patient.id}');
  }
}

FutureOr<String> _getStatisticToStr(Map<String, dynamic> data) {
  return (data['statistic'] as List<BloodSugarStatistic>).fold(
    "",
    (prevValue, value) => '$prevValue ${value.getBriefInfo(data['locale'])}\n',
  );
}

```

A.14 Код головної сторінки лікаря MainDoctorPage

lib/src/app/pages/main_doctor/main_doctor_page.dart

```

import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/dia_localizations.dart';
import 'package:provider/provider.dart';
import 'package:dia_app/src/app/pages/main_doctor/tabs/patients/patients_tab.dart';
import 'package:dia_app/src/app/pages/login/login_page.dart';

```

```

import 'package:dia_app/src/app/pages/main_doctor/tabs/doctor/doctor_tab.dart';
import 'package:dia_app/src/app/pages/main_doctor/main_doctor_page_presenter.dart';
import 'package:dia_app/src/app/routes.dart' as routes;
import 'package:dia_app/src/app/widgets/loading_layout.dart';

class MainDoctorPage extends StatefulWidget {
  static const nameRoute = routes.mainDoctor;
  static PageRoute buildPageRoute() {
    return MaterialPageRoute(
      settings: RouteSettings(name: nameRoute),
      builder: _builder,
    );
  }
  static Widget _builder(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => MainDoctorPagePresenter(),
      child: MainDoctorPage(),
    );
  }
  @override
  _MainDoctorPageState createState() => _MainDoctorPageState();
}
class _MainDoctorPageState extends State<MainDoctorPage> {
  int _selectedIndex = 1;

  DiaLocalizations get _locale => DiaLocalizations.of(context);
  MainDoctorPagePresenter get _watch => context.watch<MainDoctorPagePresenter>();
  MainDoctorPagePresenter get _read => context.read<MainDoctorPagePresenter>();

  @override
  Widget build(BuildContext context) {
    return LoadingLayout(
      isLoading: _watch.isLogOutLoading,
      child: GestureDetector(
        onTap: _resetFocusNode,
        child: Scaffold(
          appBar: AppBar(
            title: Text(_locale.appName),
            actions: [_buildAppBarAction()],
          ),
          bottomNavigationBar: BottomNavigationBar(
            currentIndex: _selectedIndex,
            onTap: _onItemTapped,
            items: <BottomNavigationBarItem>[
              BottomNavigationBarItem(
                icon: Icon(Icons.account_circle),
                label: _locale.doctor,
              ),
              BottomNavigationBarItem(
                icon: Icon(Icons.assignment_outlined),
                label: _locale.patients,
              ),
            ],
          ),
          body: _tabBuilder(),
        ),
      ),
    );
  }

  void _resetFocusNode() {
    FocusScope.of(context).requestFocus(FocusNode());
  }
}

```

```

Widget _buildAppBarAction() {
  switch (_selectedIndex) {
    case 0:
      return IconButton(
        icon: Icon(Icons.exit_to_app),
        tooltip: _locale.changeUser,
        onPressed: _logout,
      );
    default: return Container();
  }
}

Future<void> _logout() async {
  await _read.logout();
  await Navigator.of(context).pushAndRemoveUntil(
    LoginPage.buildPageRoute(),
    (route) => false,
  );
}

void _onItemTapped(int index) {
  setState(() => _selectedIndex = index);
}

Widget _tabBuilder() {
  switch (_selectedIndex) {
    case 0: return DoctorTab();
    case 1: return PatientsTab();
    default: return Container();
  }
}
}

```

A.15 Код сценарію використання для лікаря MainPagePresenter

lib/src/app/pages/main_doctor /main_doctor page_presenter.dart

```

import 'package:dia_app/src/domain/entities/user.dart';
import 'package:dia_app/src/domain/repositories_contracts/auth_repository.dart';
import 'package:dia_app/src/domain/repositories_contracts/user_repository.dart';
import 'package:flutter/foundation.dart';
import 'package:get_it/get_it.dart';

class MainDoctorPagePresenter with ChangeNotifier {
  final AuthRepository _authRepository;
  final DoctorRepository _doctorRepository;

  Doctor _doctor;
  List<Patient> _patients;
  bool _isLogoutLoading = false;
  bool _isDoctorLoading = false;
  bool _isPatientsLoading = false;

  Doctor get doctor => _doctor;
  List<Patient> get patients => _patients;
  bool get isLogoutLoading => _isLogoutLoading;
  bool get isDoctorLoading => _isDoctorLoading;
  bool get isPatientsLoading => _isPatientsLoading;

  MainDoctorPagePresenter()
    : _authRepository = GetIt.I.get<AuthRepository>(),
      _doctorRepository = GetIt.I.get<UserRepository>() {

```

```

    init();
  }
  Future<void> init() async {
    _isDoctorLoading = true;
    _isPatientsLoading = true;
    notifyListeners();
    final results = await Future.wait([
      _doctorRepository.fetchDoctor(),
      _doctorRepository.fetchPatients(),
    ]);
    _doctor = results[0];
    _patients = results[1];
    _isDoctorLoading = false;
    _isPatientsLoading = false;
    notifyListeners();
  }
  Future<void> addPatient(String patientId) async {
    _isPatientsLoading = true;
    notifyListeners();
    await _doctorRepository.addPatient(patientId);
    _patients = await _doctorRepository.fetchPatients();
    _isPatientsLoading = false;
    notifyListeners();
  }
  Future<void> logOut() async {
    _isLogOutLoading = true;
    notifyListeners();
    await _authRepository.logOut();
    _isLogOutLoading = false;
    notifyListeners();
  }
}

```

A.16 Код для тестування сутності **Statistic**, дочірнього класу

BloodSugarStatistic та методів `test/src/domain/entities/statistic_test.dart`

```

import 'dart:ui';
import 'package:dia_app/src/domain/entities/statistic.dart';
import 'package:flutter_test/flutter_test.dart';

main() {
  final bloodSugarStatistic = BloodSugarStatistic(
    bloodSugar: 5.0,
    dateTimeOfMeasure: DateTime(2021, 10, 11),
  );
  final bloodSugarStatisticList = [
    BloodSugarStatistic(
      bloodSugar: 3.0,
      dateTimeOfMeasure: DateTime(2021, 10, 10),
    ),
    BloodSugarStatistic(
      bloodSugar: 5.0,
      dateTimeOfMeasure: DateTime(2021, 10, 11),
    ),
    BloodSugarStatistic(
      bloodSugar: 6.0,
      dateTimeOfMeasure: DateTime(2021, 10, 12),
    ),
  ];
  test(
    'Statistic type : BloodSugarStatistic is Statistic',

```

```

    () {
      expect(bloodSugarStatistic, isA<Statistic>());
    },
  );
  test(
    'Statistic : DateTime Formatting',
    () {
      final expected = '11.10 00:00';
      expect(bloodSugarStatistic.getSimpleDateTime(), expected);
    },
  );
  test(
    'BloodSugarStatistic : diagnosis',
    () {
      expect(bloodSugarStatisticList[0].diagnosis, BloodSugarDiagnosis.hypoglycemia);
      expect(bloodSugarStatisticList[1].diagnosis, BloodSugarDiagnosis.normal);
      expect(bloodSugarStatisticList[2].diagnosis, BloodSugarDiagnosis.hyperglycemia);
    },
  );
  test(
    'BloodSugarStatistic : color of item',
    () {
      expect(bloodSugarStatisticList[0].color, Color(0xFFA0C4FF));
      expect(bloodSugarStatisticList[1].color, Color(0xFFCAFFBF));
      expect(bloodSugarStatisticList[2].color, Color(0xFFffadad));
    },
  );
}

```

A.17 Код для тестування сутності User та дочірніх класів Patient і

Doctor test/src/domain/entities/models/user_test.dart

```

import 'package:dia_app/src/domain/entities/user.dart';
import 'package:flutter_test/flutter_test.dart';

```

```

main() {
  final patient = Patient(
    id: 'Patient987654321',
    fullName: 'Patient',
    email: 'patient@gmail.com',
    docEmail: 'doctor@gmail.com',
  );
  final doctor = Doctor(
    id: 'Doctor987654321',
    fullName: 'Doctor',
    email: 'doctor@gmail.com',
    patientIds: [
      'Patient987654321',
    ],
  );
  test(
    'User type : Patient is User',
    () {
      expect(patient, isA<User>());
    },
  );
  test(
    'User type : Doctor is User',
    () {
      expect(doctor, isA<User>());
    },
  );
}

```

```
);
}
```

A.18 Код для тестування методів репозиторію AuthRepository

test/src/data/repositories_implementations/auth_repository_test.dart

```
import 'package:dia_app/src/data/mock_repositories/mock_auth_repository.dart';
import 'package:dia_app/src/data/mock_repositories/mock_user_repository.dart';
import 'package:dia_app/src/domain/entities/user.dart';
import 'package:dia_app/src/domain/repositories_contracts/auth_repository.dart';
import 'package:dia_app/src/domain/repositories_contracts/user_repository.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:get_it/get_it.dart';
```

```
main() {
  AuthRepository _authRepository;
  setUp(() {
    _authRepository = MockAuthRepository();
    GetIt.I.registerSingleton<UserRepositoryFactory>(
      MockUserRepositoryFactory(),
    );
  });
  tearDown(() {
    GetIt.I.unregister<UserRepositoryFactory>();
  });
  group('AuthRepository', () {
    test('init', () async {
      expect(UserType.Patient, await _authRepository.init());
    });
    test('isLoggedIn', () async {
      expect(true, await _authRepository.isLoggedIn);
    });
    test(
      'userType',
      () async {
        expect(UserType.Patient, equals(await _authRepository.userType));
      },
    );
  });
}
```

A.19 Код для тестування фабрики UserRepositoryFactory, репозиторію UserRepository та дочірніх реалізацій

test/src/data/repositories_implementations/user_repository_test.dart

```
import 'package:dia_app/src/data/mock_repositories/mock_user_repository.dart';
import 'package:dia_app/src/domain/entities/statistic.dart';
import 'package:dia_app/src/domain/entities/user.dart';
import 'package:dia_app/src/domain/repositories_contracts/user_repository.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter_test/flutter_test.dart';
```

```
main() {
  UserRepositoryFactory _userRepositoryFactory;
  PatientRepository _patientRepository;
  DoctorRepository _doctorRepository;

  setUp(() {
    _userRepositoryFactory = MockUserRepositoryFactory();
```

```

    _patientRepository = MockPatientRepository();
    _doctorRepository = MockDoctorRepository();
  });
  group('UserRepositoryFactory', () {
    test('create PatientRepository', () {
      final repository = _userRepositoryFactory.createUserRepository(UserType.Patient);
      expect(repository, isA<PatientRepository>());
    });
    test('create DoctorRepository', () {
      final repository = _userRepositoryFactory.createUserRepository(UserType.Doctor);
      expect(repository, isA<DoctorRepository>());
    });
  });
  group('PatientRepository', () {
    test('fetchBloodSugarStatistic', () async {
      final statistic = await _patientRepository.fetchBloodSugarStatistic();
      expect(
        true,
        listEquals(
          statistic,
          [
            BloodSugarStatistic(bloodSugar: 3.0, dateTimeOfMeasure: DateTime(2021, 10, 10)),
            BloodSugarStatistic(bloodSugar: 5.0, dateTimeOfMeasure: DateTime(2021, 10, 11)),
            BloodSugarStatistic(bloodSugar: 6.0, dateTimeOfMeasure: DateTime(2021, 10, 12)),
          ],
        ),
      );
    });
    test(
      'fetchCurrentPatient',
      () async {
        final patient = await _patientRepository.fetchCurrentPatient();
        expect(
          patient,
          equals(
            Patient(
              id: 'Patient987654321',
              fullName: 'Patient',
              email: 'patient@gmail.com',
              docEmail: 'doctor@gmail.com',
            ),
          ),
        );
      },
    );
  });
  group('DoctorRepository', () {
    test('fetchPatients', () async {
      final statistic = await _doctorRepository.fetchPatients();
      expect(
        statistic,
        equals(
          [
            Patient(
              id: 'Patient987654321',
              fullName: 'Patient',
              email: 'patient@gmail.com',
              docEmail: 'doctor@gmail.com',
            ),
          ],
        ),
      );
    });
  });

```

```

});
group('UserRepository', () {
  test('fetchDoctor', () async {
    final patient = await _patientRepository.fetchDoctor();
    expect(
      patient,
      equals(
        Doctor(
          id: 'Doctor987654321',
          fullName: 'Doctor',
          email: 'doctor@gmail.com',
          patientIds: ['Patient987654321'],
        ),
      ),
    );
  });
});
test('fetchBloodSugarStatisticByPatientId', () async {
  final statistic = await _doctorRepository.fetchBloodSugarStatisticByPatientId('Patient987654321');
  expect(
    true,
    listEquals(
      statistic,
      [
        BloodSugarStatistic(bloodSugar: 3.0, dateTimeOfMeasure: DateTime(2021, 10, 10)),
        BloodSugarStatistic(bloodSugar: 5.0, dateTimeOfMeasure: DateTime(2021, 10, 11)),
        BloodSugarStatistic(bloodSugar: 6.0, dateTimeOfMeasure: DateTime(2021, 10, 12)),
      ],
    ),
  );
});
test('fetchPatientById', () async {
  final patient = await _patientRepository.fetchPatientById('Patient987654321');
  expect(
    patient,
    equals(
      Patient(
        id: 'Patient987654321',
        fullName: 'Patient',
        email: 'patient@gmail.com',
        docEmail: 'doctor@gmail.com',
      ),
    ),
  );
});
});
}

```

ДОДАТОК Б
СЛАЙДИ ПРЕЗЕНТАЦІЇ

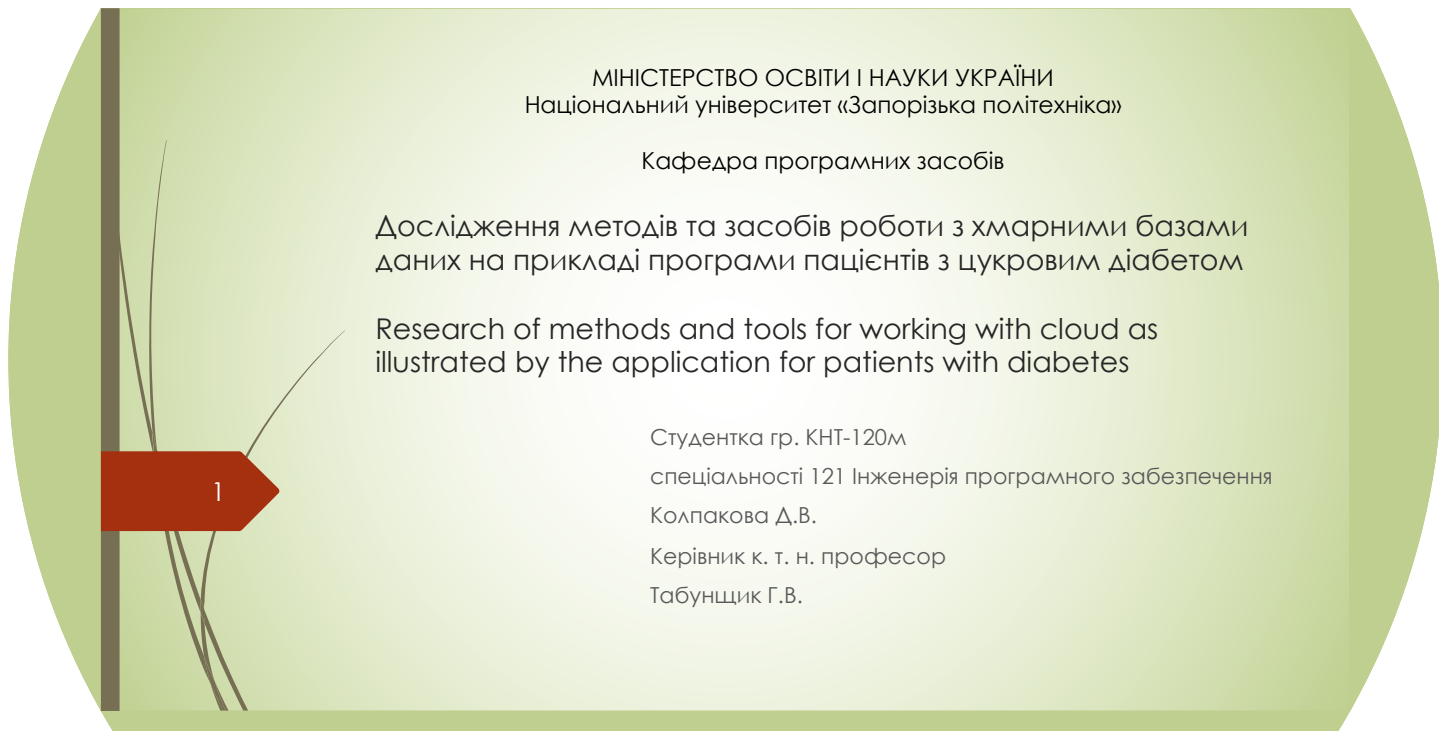


Рисунок Б.1 – Титульний слайд

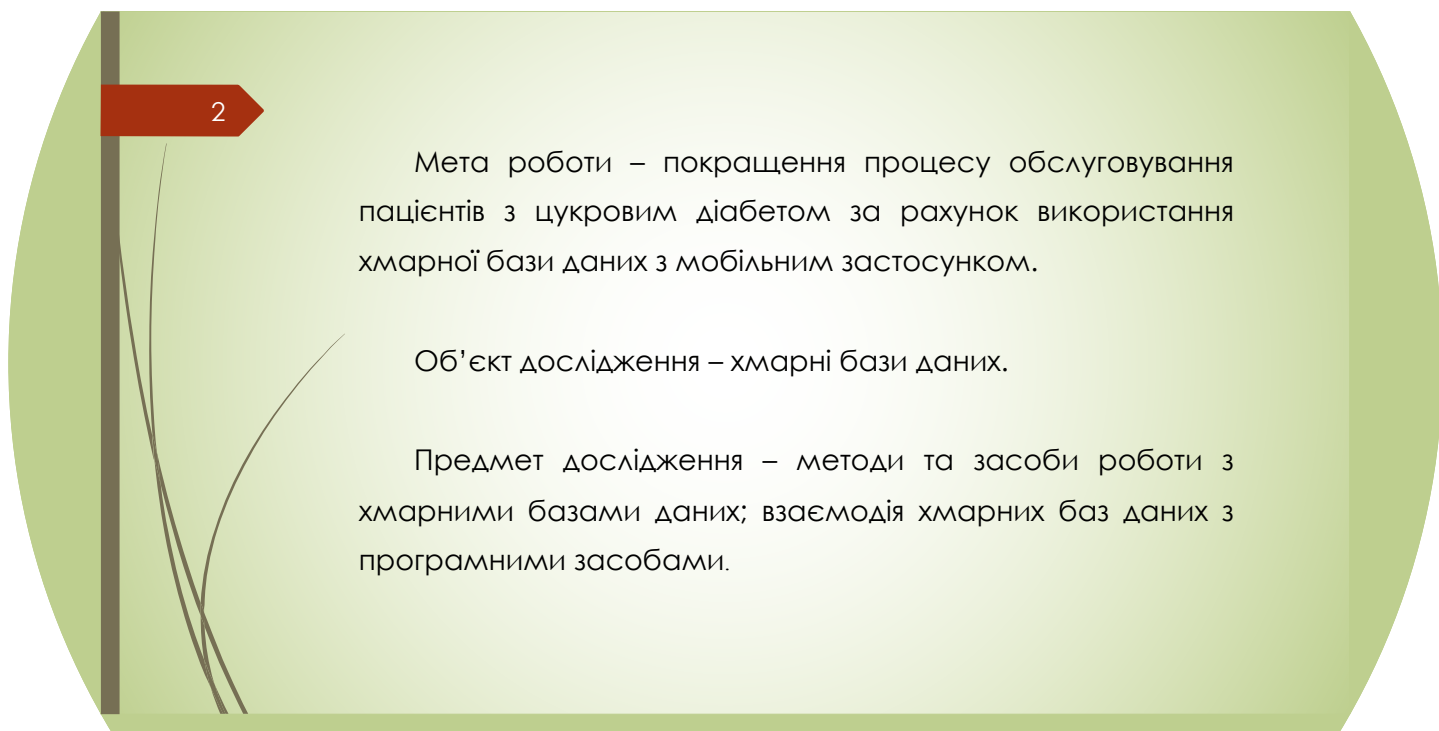


Рисунок Б.2 – Мета роботи, об'єкт та предмет дослідження

3

Порівняльний аналіз серверних постачальників послуг

Показники	Firebase	AWS Amplify	Backendless
Рік заснування	2011	2017	2012
Бази даних	Cloud Firestore та Realtime Database (NoSQL)	SQL і NoSQL від відомих систем управління управління БД	SQL і NoSQL власної розробки, зі зручним UI
Рівень складності	Середній, наявність активній спільноті розробників та детальна документація спрощує зайомство з платформою	Високий, складна документація яка постійно доповнюється інформацією про нові функції та сервіси	Низький, знайомство з можливостями платформи відбувається у вигляді гри
Тарифні плани	Spark: безкоштовний, з квотами використання сервісів; Blaze: сплата лише за ті, послуги, які використовуються	Free tier: враховує квоти та є безкоштовним лише перші 12 місяців; Pay as you go: активується після після року використання та сплата лише за ті, послуги, які використовуються	Springboard: безкоштовна але дуже обмежена; Cloud 9: для застосунків з невеликою аудиторією; Cloud 99: для великих підприємств

Рисунок Б.3 – Аналіз серверних постачальників послуг

4

Порівняльний аналіз програмних продуктів для хворих на діабет

Показники	MySugr	GlucoseTracker	OneTouch Reveal
Зручність та простота інтерфейсу користувача	+	-	+
Можливість відображення графіків	-	+	+
Повідомлення про наступну ін'єкцію	+	+	-
Безкоштовна	-	-	-
Калькулятор інсуліну	+	-	+
Офлайн режим	-	-	+
Наявність версій для різних ОС	+	-	+
Стабільність роботи	-	-	-

Рисунок Б.4 – Аналіз програмних продуктів для хворих на діабет

Бізнес-модель “Канвас”

5

Ефективність			Цінність	
8. Ключові партнери	7. Ключові види діяльності	2. Ціннісна пропозиція	4. Взаємовідносини з клієнтом	1. Споживчий сегмент
Приватні лікарні та клініки; державне міністерство охорони здоров'я.	Для пацієнтів та лікарів: комунікація та постійний доступ до інформації про стан пацієнта. Для держави: збір статистичних даних про стан проблеми в країні.	Зручність та простота використання; лікарі мають постійний доступ до актуальних даних про пацієнтів.	Застосунок безкоштовний для пацієнтів, але є домовленості з приватними лікарнями або лікарями.	Пацієнти з діагнозом цукровий діабет у віці 20-50 років.
	6. Ключові ресурси		3. Канали збуту	Лікарні та лікарі, яким необхідно знати про стан пацієнта онлайн.
	Команда розробників з 2-3 чол., VaaS, офісне приміщення, інвестиції лікарень або держбюджету.		Брошури у лікарнях; реклама та посилання на сайтах приватних клінік та на сайті МОЗ.	
9. Структура витрат		5. Потоки надходження доходів		
Розробка та технічна підтримка; реклама; VaaS (сплата за використання ресурсів платформи).		Від приватних лікарень (лікарів) або державного бюджету, як застосунок який надає актуальні дані про пацієнтів.		

Рисунок Б.5 – Бізнес-модель “Канвас”

6

Схематичне представлення теорем CAP та PACELC

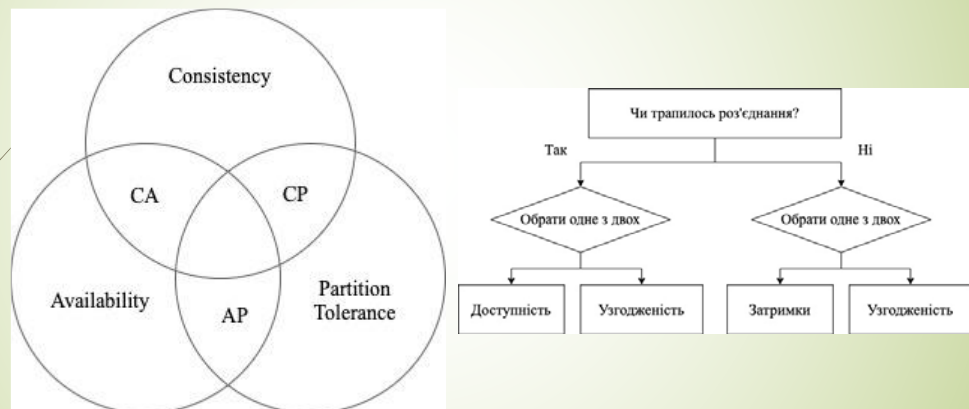


Рисунок Б.6 – Представлення теорем CAP та PACELC

7

Схематичне представлення архітектури проєкту

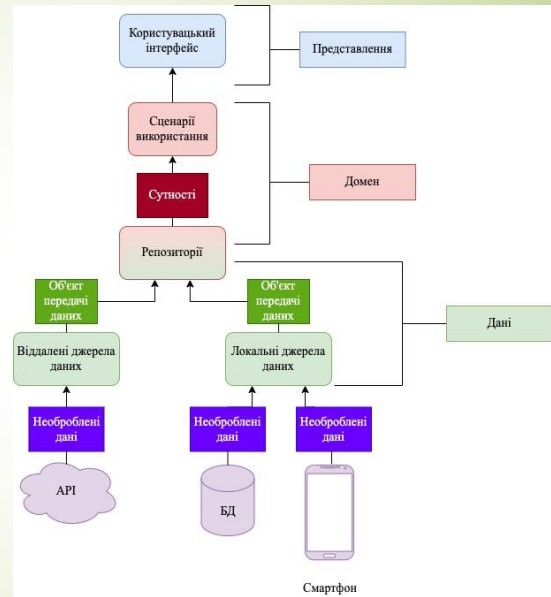


Рисунок Б.7 – Схема архітектури проєкту

8

Діаграма прецедентів

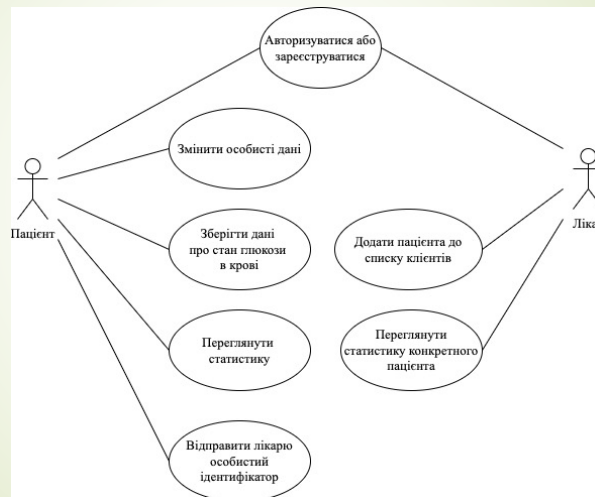


Рисунок Б.8 – Діаграма прецедентів

9

Діаграма класів адаптивного архітектурного шаблону

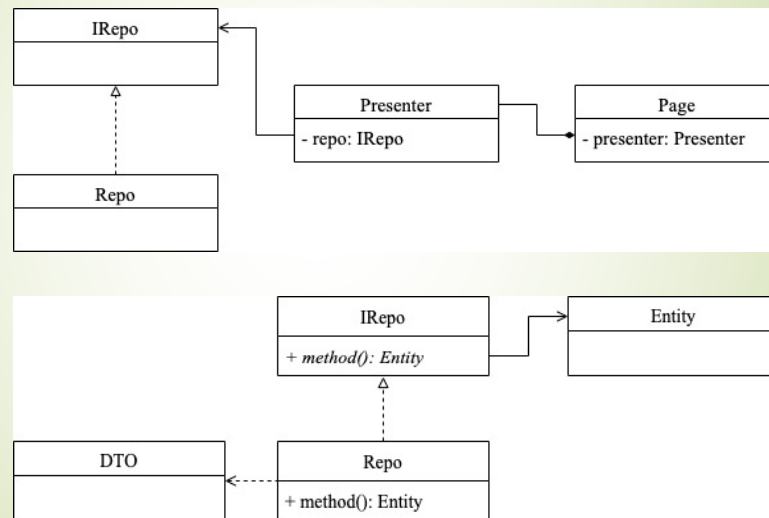


Рисунок Б.9 – Діаграма класів архітектурного шаблону

10

Діаграма класів сутностей

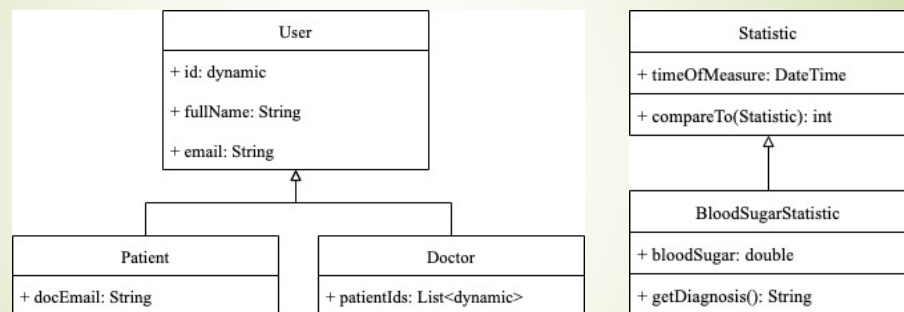


Рисунок Б.10 – Діаграма класів сутностей

11

Діаграма класів репозиторіїв з використанням патерну “Фабрика”

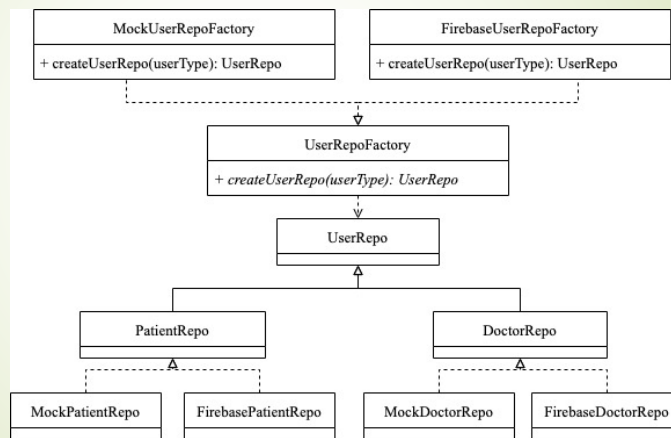


Рисунок Б.11 – Діаграма класів репозиторіїв з використанням патерну “Фабрика”

12

Діаграма взаємодії класів авторизації та фабричного класу

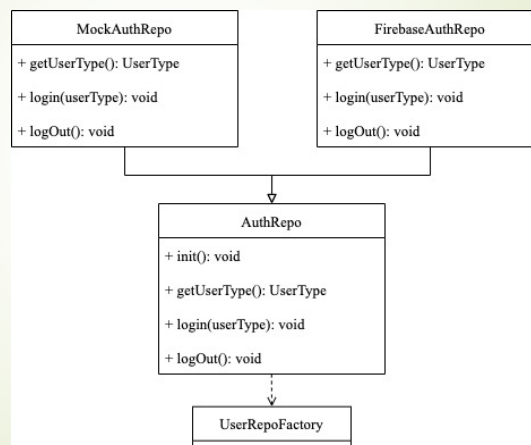
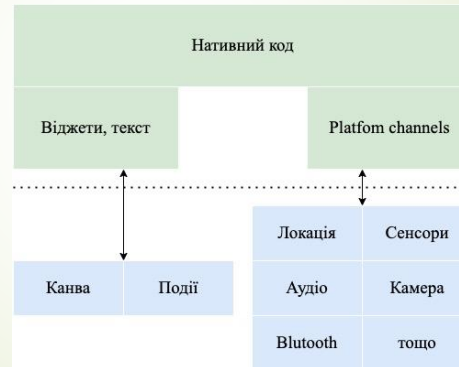


Рисунок Б.12 – Діаграма взаємодії класів авторизації та фабричного класу

13

Мобільний застосунок вирішено розробити з використанням мови програмування Dart, програмного каркасу Flutter, додаткових бібліотек, для спрощення реалізації програмного рішення, та середовища розробки Android Studio.

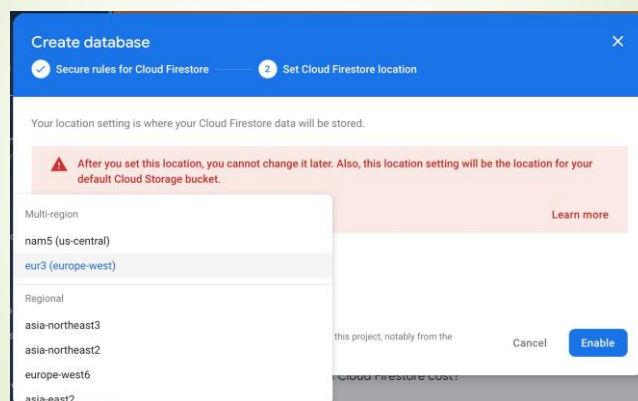


Схематичне зображення роботи застосунку написаному на Flutter

Рисунок Б.13 – Вибір засобів та інструментів розробки

14

Налаштування серверного постачальника послуг

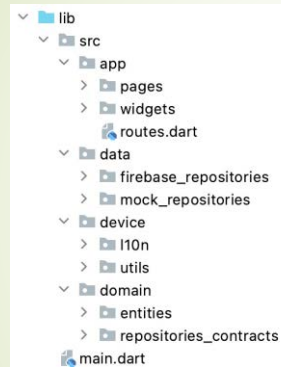


Налаштування місцезнаходження БД

Рисунок Б.14 – Налаштування БД

15

Розробка програмного продукту



```
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  GetIt.I
    ..registerSingleton<GoogleService>(GoogleService())
    ..registerSingleton<StoreInteractor>(StoreInteractor())
    ..registerSingleton<AuthRepository>(FirebaseAuthRepository())
    ..registerSingleton<UserRepositoryFactory>(FirebaseUserRepositoryFactory());
  final userType = await GetIt.I<AuthRepository>().init();
  runApp(MyApp(userType: userType));
}
```

Рисунок Б.15 – Розробка застосунку

16

Тестування Cloud Firestore

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /patients/{userId}/{document=**} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
  }
}
```

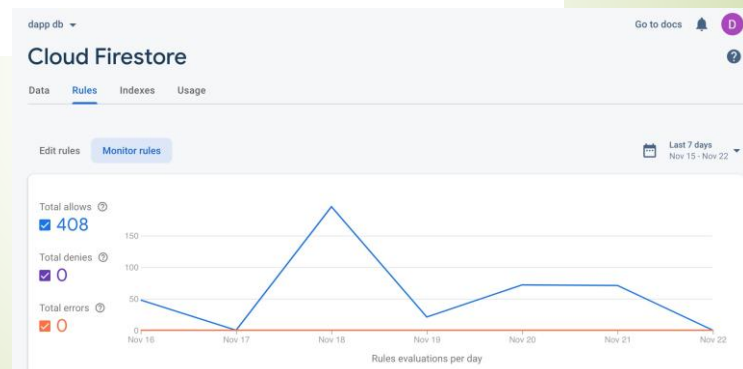


Рисунок Б.16 – Тестування Cloud Firestore

17

Дані про використання ресурсів Cloud Firestore

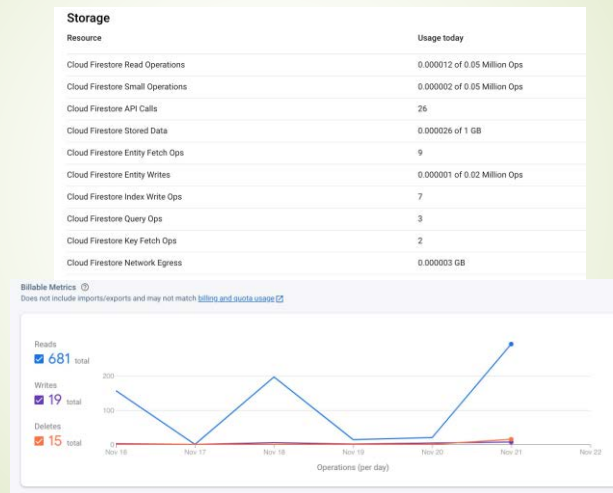


Рисунок Б.17 – Дані про використання квот Cloud Firestore

18

Інтерфейс користувача

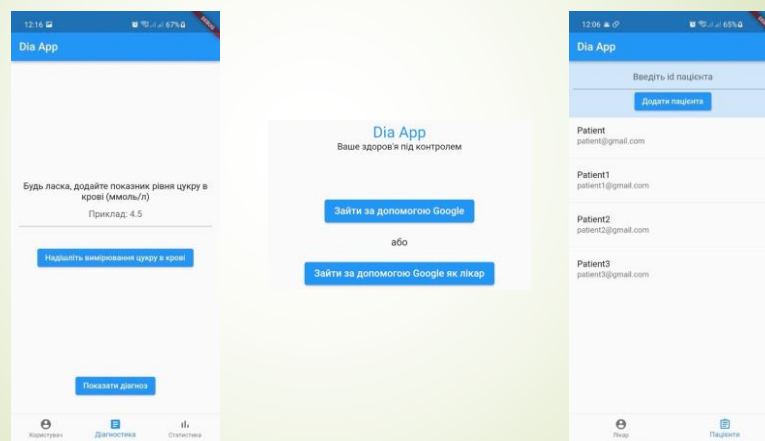


Рисунок Б.18 – Інтерфейс користувача

19

Модульне тестування програми

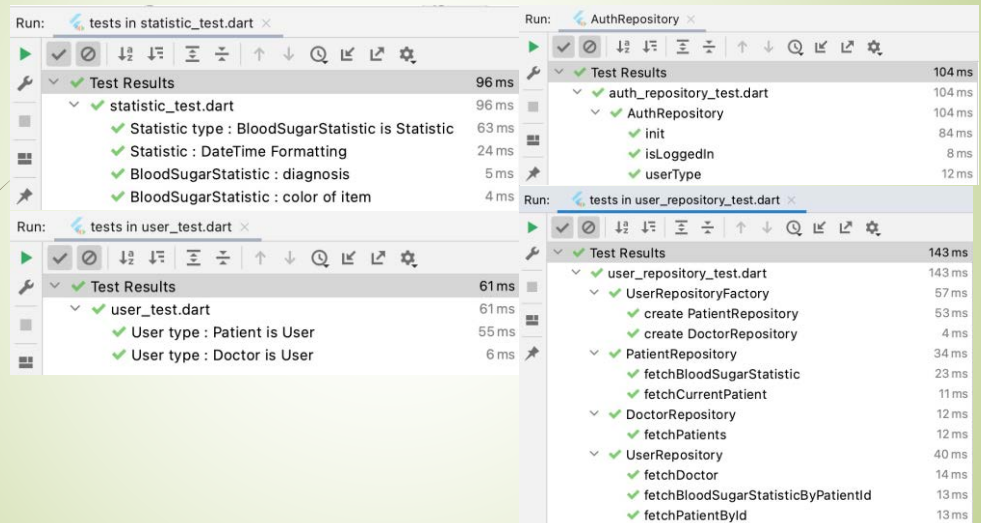


Рисунок Б.19 – Модульне тестування програми

20

Висновки

Результатом роботи є застосунок для пацієнтів з цукровим діабетом та лікарів, який використовує Firebase у якості серверу.

Для досягнення поставленої мети у роботі, було вирішено завдання, серед яких:

- 1) Проаналізовано існуючі бекенд як сервіси. Для розробки застосунку було обрано Firebase у зв'язку з тим, що ця платформа надійніша за інші рішення та має більше безкоштовних сервісів.
- 2) Враховуючи принципи чистої архітектури, розроблено детальну схему архітектури застосунку, запропоновано методи впровадження API платформи у проект.
- 3) Описано бізнес-логіку програмного продукту. Розроблено архітектуру проекту, з урахуванням запропонованих методів, створено архітектурний шаблон. Детально описано взаємодію класів та компонентів проекту.
- 4) Протестовано сценарії використання Cloud Firestore, проведено ручне тестування взаємодії застосунку з платформою, продемонстровано можливості інтерфейсу розробленого застосунку, проведено модульне тестування. За результатами тестування помилок не виявлено.
- 5) Часткові результати даної кваліфікаційної роботи, було подано як тезу на науково-практичну конференцію викладачів, науковців, молодих учених, аспірантів та студентів "Тиждень науки-2021".

Рисунок Б.20 – Висновки



Рисунок Б.21 – Останній слайд